Entwicklung eines Konzeptes zur Wissensextraktion durch den GETESS–Gatherer

Diplomarbeit
Universität Rostock, Fachbereich Informatik

vorgelegt von
Ilvio Bruder
geboren am 20.12.1975 in Bützow

Betreuer: Prof. Dr. Andreas Heuer
Dr. Ing. Antje Düsterhöft

Abgabedatum: Rostock, den 14. Juli 2000

Zusammenfassung

In dieser Arbeit werden Techniken zur Wissensextraktion aus Web–Dokumenten betrachtet. Diese Extraktion beinhaltet mehrere Analyseschritte, um strukturelle, linguistische, domänenspezifische und inhaltliche Informationen zu erhalten.
Es wird insbesondere betrachtet, wie die extrahierten Daten zu einer möglichst genauen Information über die betreffende Web–Seite kombiniert werden können. Das Ergebnis besteht im wesentlichen aus einem sogenannten abstract, welches die resultierenden Informationen in geeigneter Weise darstellt.
Die hier entwickelten Analysewerkzeuge werden im GETESS–Projekt zur Unterstützung der Wissensextraktion umgesetzt und evaluiert.

Abstract

This paper illustrates techniques of knowledge extraction from web documents. This extraction contains several analysis steps to get structural, linguistic, domain specific and content based information.
Especially, it will be shown how the extracted data could be combined to an information about the web site as precise as possible. The result consists essentially of a so called abstract. This abstract represents the resulted information in the right way.
The here evolved analysis tools will be implemented and evaluated in the GETESS project to support the knowledge extraction.

CR–Klassifikation

H.3.3	Information Search and Retrieval	
I.2.7	Natural Language Processing	(Text analysis)
I.5	Pattern Recognition	
I.7	Document and Text Processing	
I.7.2	Document Preparation	
I.7.5	Document Capture	(Document analysis)

Keywords

Internet–Suchmaschinen, semistrukturierte Daten, Informationsextraktion, wissensbasierte Systeme, Web–Datenanalyse, Strukturanalyse, domänenspezifische linguistische Analyse

Inhaltsverzeichnis

Kapitel 1

Einleitung

In vielen Forschungsgebieten wird an Wissensextraktion, Datenspeicherung und Datenzugriff gearbeitet. Bei diesen Forschungsarbeiten entstehen viele einzelne, interessante Konzepte für eine Vielzahl von auftretenden Problemen. Ein Zusammenführen solcher Konzepte aus verschiedenen Forschungsgebieten ist eine sinnvolle, aber nicht einfache Aufgabe.
Das Projekt GETESS[1] ist ein Projekt, in dem eine Internetsuchmaschine mit erweiterter Funktionalität entsteht und vier Projektpartner aus sehr verschiedenen Forschungsgebieten, Datenbanken, Computerlinguistik und Knowledge Engineering, zusammenarbeiten.
Diese Arbeit befaßt sich mit den Möglichkeiten der Wissensextraktion im Projekt GETESS. Dabei steht das Zusammenführen verschiedener Konzepte aus unterschiedlichen Forschungsgebieten im Vordergrund.
Im folgenden werden das Projekt GETESS vorgestellt und die Thematik dieser Arbeit genauer eingeordnet.

1.1 Suchmaschinen im Internet

Bevor das Projekt GETESS als Internet–Suchmaschine vorgestellt wird, gibt dieser Abschnitt eine Einführung in die Problematik der Suchmaschinen.
Im Internet sind heute viele Suchmaschinen unterschiedlicher Qualität zu finden. Die Architektur und die Vorgehensweise ähneln sich allerdings sehr, sei es Altavista, Lycos oder Webcrawler.
Nach [BZW98] sind Suchmaschinen im Internet eine Anwendung intelligenter Softwareagenten. [BZW98] gibt eine Klassifizierung (Entwicklungsstufen) der Suchmaschinen und stellt deren Konzepte und Architektur vor. Die drei wesentlichen Klassen, einfache Suchmaschinen, Meta Suchmaschinen und personalisierte Suchmaschinen, werden im folgenden nach Reihenfolge der Entwicklungsstufe beschrieben.

Es gibt **einfache Suchmaschinen**, die im wesentlichen auf drei Konzepten beruhen. Zum einen ist es die Erfassung von Informationen durch Link–Verfolgung mittels sogenannter robots (Suchagenten, die durch die Verfolgung von Links eine Menge von Dokumenten sammeln). Das zweite Konzept umfaßt die Indizierung der Informationen und die Speicherung in einer Speicherkomponente. Die Indizierung ist eine sehr einfache und effektive Art den Inhalt zu erschließen. Dabei werden lediglich bestimmte Terme herausgefiltert und ein Index über die Terme gebildet. Eine semantische Analyse erfolgt nicht. Das dritte Konzept bildet den Zugriff auf die Daten über nutzerspezifische Suchanfragen. Dabei wird die Suchanfrage mit den Repräsentatoren, den Termen, jedes Dokumentes verglichen und so mögliche, relevante Dokumente geliefert. Viele Suchmaschinen bieten die Ergebnisliste mit einem Ranking an.
Die Architektur einfacher Suchmaschinen besteht nach [BZW98] aus vier wesentlichen Komponenten. Der Such–Administrator übernimmt die Kontrolle der Informationserfassung und führt die Indizierung durch. In der zweiten Komponente sind die Agenten für das Auffinden der zu indizierenden Dokumente verantwortlich. Eine Datenbank übernimmt das Speichern des Indexes. Und ein

[1] GETESS – GErman Text Exploitation and Search System.

Anfrageserver stellt den Zugriff über eine Nutzerschnittstelle, meist ein HTML–Formular, bereit.

Die nächste Art der Suchmaschinen stellen die **Meta Suchmaschinen** dar. Beispiele für diese Suchmaschinen sind Metacrawler oder MetaGer. Meta Suchmaschinen versuchen qualitativ hochwertige Ergebnisse durch parallele Abfrage der Daten von einfachen Suchmaschinen zu erzielen. Diese Klasse von Internetsuchmaschinen beruht auf zwei wesentliche Konzepte. Zum einen wird die Suchanfrage geglättet und an die Schnittstellen der einfachen Suchmaschinen adaptiert. Das heißt, die Anfrage wird an die Gegebenheiten bzw. Kriterien der einfachen Suchmaschinen durch Modifizierung angepaßt und vermittelt. Das zweite Konzept betrifft das Zusammenführen der verschiedenen Ergebnisse der angesprochenen, einfachen Suchmaschinen. Die aggregierte Ergebnismenge wird analysiert, bewertet und dann präsentiert.
Die Architektur hat ähnlich den einfachen Suchmaschinen vier Komponenten. Ein Such–Administrator übernimmt die Kommunikation mit den einfachen Suchmaschinen, analysiert und bewertet die Ergebnisse und bedient die Anfrageschnittstelle. Die modulare Plattform besteht aus den einzelnen, einfachen Suchmaschinen mit der zentralen Aufgabe diese zusammenzuführen. Die Agenten sind für das Auffinden der referenzierten Dokumente verantwortlich und die Anfrageschnittstelle definiert die Suchanfrage und präsentiert die Ergebnisse.

Eine Erweiterung von Meta Suchmaschinen stellen die **personalisierten Suchmaschinen** dar. Diese basieren auf der Idee der Personalisierung der Ergebnismengen, um die speziellen Interessengebiete des Nutzers zu unterstützen.

GETESS, als Suchmaschine mit wesentlichen Erweiterungen, ist nicht in diese Hierarchie nach [BZW98] einzuordnen. GETESS versucht durch intelligente, natürlichsprachliche und wissensbasierte Konzepte die Informationsextraktion, die Speicherung und die Nutzerkommunikation zu verbessern. Der Suchraum wird durch Auswahl eines Themengebietes bzw. Domäne eingegrenzt. Nach [BZW98] ist dies eine Verschiebung der Suchmaschinen zu intelligenteren Agentensystemen, da das Wissenspotential erhöht wird.

Im Internet gibt es sehr viele Internet–Suchmaschinen aller drei, oben vorgestellten Kategorien. Einen Überblick über Internet–Suchmaschinen und einige Aspekte der Bewertung von Suchmaschinen sind in [sew00], [sea00a] und [sea00b] zu finden.

1.2 Das Projekt GETESS

GETESS ([BPW+98] und [MPP+99]) ist ein dreijähriges Projekt des Bundesministerium für Bildung und Forschung (kurz BMBF) zur Integration von computerlinguistischen Techniken, wissensbasierten Techniken, Datenbanktechniken und Dialogtechniken in eine Suchmaschine. Ziel des Projektes ist die Entwicklung eines intelligenten Werkzeugs zur Bereitstellung und Recherche von Informationen im Internet. Ein derartiges Konzept einer Internet–Suchmaschine mit erweiterter Funktionalität wird weitreichende Verbesserungen für den Nutzer mitbringen.

Folgende Anforderungen (aus [BPW+98]) werden in diesem Projekt eine wesentliche Rolle spielen:

1. Es soll ein einfach zu bedienendes System sein.
2. Der Nutzer soll die Möglichkeit haben, natürlichsprachliche Anfragen zu stellen.
3. Der Nutzer soll zur besseren Kommunikation in einer Ontologie navigieren können.
4. Die Präsentation der Suchergebnisse soll in geeigneter und nachvollziehbarer Form erfolgen.
5. Moderne und attraktive Multimediatechniken kommen zum Einsatz.

1.2.1 Projektpartner und Aufgaben

An diesem Projekt sind vier, sehr unterschiedliche Projektpartner aus der Bundesrepublik beteiligt.

Zu den Partnern werden nachfolgend deren Aufgaben innerhalb des Projekts genannt und beschrieben. Die Zusammenhänge der Aufgaben werden unten mit einer Architekturskizze von GETESS gezeigt.

Die **GECKO mbH Rostock** mit Geschäftsführer Siegfried Melzig ist eine Software– und Provider–Firma aus Rostock. Bei der GECKO mbH werden eine Dialogkomponente zur Interaktion mit dem Benutzer entwickelt und eine Common Knowledge Base[2] (CKB) konzipiert und realisiert.
Das Deutsche Forschungsinstitut für Künstliche Intelligenz (kurz **DFKI**) in **Saarbrücken** mit Prof. Hans Uszkoreit wird mit computerlinguistischen Techniken aus den Informationsquellen wichtige Daten zur Weiterverarbeitung extrahieren und eine multilinguale Verarbeitung natürlicher Sprache in das Projekt einbringen.
Die Fachgruppe Wissensmodellierung und wissensbasierte Systeme (kurz **AIFB Uni Karlsruhe**) unter Prof. Rudi Studer an der Universität Karlsruhe ist für den Entwurf und Realisierung einer Domänen–Ontologie verantwortlich und stellt den Zusammenhang zwischen linguistischen und ontologischen Aspekten her.
Als vierter Partner ist der Lehrstuhl Datenbank- und Informationssysteme an der Universität Rostock (**DBIS Uni Rostock**) unter Prof. Andreas Heuer für die Entwicklung eines Internet–Sammelagenten (Gatherer) für das System, die Speicherung der Daten in eine Datenbank und den Zugriff auf die Daten der verschiedenen Datenquellen verantwortlich.

1.2.2 GETESS-Architektur

In der Abbildung 1.1 wird die Komponentenarchitektur von GETESS gezeigt. Die Komponenten des Projektes sind in der Skizze verallgemeinert und in größere Abschnitte zusammengefaßt dargestellt. Aufgrund der Übersichtlichkeit sind keine Verbindungen der Komponenten in der Skizze enthalten. Die Beziehungen werden durch die Anordnung der Komponenten ausgedrückt.

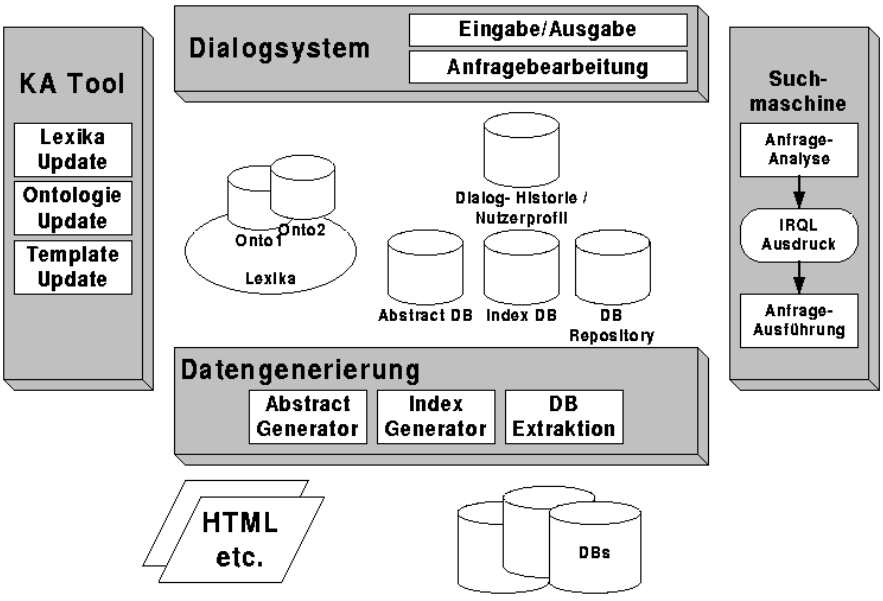

Abbildung 1.1: Architekturskizze der Komponenten von GETESS

[2]Common Knowledge Base — Wissensbank für Allgemeinwissen.

Umrahmt durch die Hauptkomponenten werden in der Mitte der Architekturskizze die Daten und die Datenhaltung dargestellt. Zum einen sind hier die Lexika für die natürlichsprachlichen Erkennungsmodule und die Ontologien für das domänenspezifische Wissen zu finden. Zum anderen liegen hier die eingesammelten und aufbereiteten Daten aus dem Internet als abstracts[3] (extrahierte domänenspezifische Daten eines Dokumentes in zusammengefaßter Form), als indizierte Daten oder als Datenbank–Repository (für externe Internetdatenbanken) vor. Außerdem werden hier weitere interne Daten, wie z.B. die Benutzerprofile, vorgehalten.

Auf der linken Seite sind die Knowledge Aquisition–Tools (kurz KA–Tools) zu finden. Diese Tools dienen zum Aufbau, Update und Löschen von Konzepten in den domänenspezifischen Datenquellen (Ontologien, Lexika).

Unterhalb der Datenhaltung sind die Komponenten zusammengefaßt, die die Informationen der Dokumente und anderer Datenquellen aus dem Internet extrahieren, aufbereiten und geeignet der Speicherung übergeben. Beim Aufbereiten der Daten liegt das Hauptaugenmerk auf der Abstract–Generierung, da hier mittels natürlichsprachlichen Parsern und Domänenwissen nicht nur lexikalisch und syntaktisch, sondern auch semantisch die Datenquellen erfaßt werden können.

Auf der rechten Seite ist der Block "Suchmaschine" angeordnet. Mit dem Begriff "Suchmaschine" ist an dieser Stelle der eigentliche Zugriff auf die extrahierten Daten gemeint. Dies beinhaltet die Anfrageanalyse, die Umwandlung in eine geeignete interne Anfragesprache (sie sollte alle Datenquellen und deren spezifische Funktionen effizient ausnutzen können) und die Ausführung der Anfrage.

Oben angeordnet befindet sich das Dialogsystem. Darüber wird der Kontakt zum Nutzer (Anfragen–Steller)[4] der GETESS–Suchmaschine realisiert. Das Dialogsystem versucht natürlichsprachlich mit dem Nutzer zu kommunizieren. Dabei werden alle internen Datenbanken benutzt, um das Informationsbedürfnis des Nutzers zu stillen. Durch Dialoge kann der Nutzer zu seiner Information geführt werden, falls das erste Suchergebnis den Nutzer noch nicht zufrieden gestellt hat.

1.3 Motivation und Einordnung in GETESS

Nachdem eine Einordnung dieser Arbeit innerhalb der Aufgabenpakete der Universität Rostock erfolgt ist, werden die Problemstellungen dieser Arbeit motiviert.

1.3.1 Einordnung in GETESS

Die GETESS–Aufgabenpakete der Universität Rostock umfassen das Gathering der Daten, das Speichern der aufbereiteten Daten und den Zugriff auf die Daten.
Das Gathering geschieht durch einen Sammelagenten bzw. Gatherer. Der Gatherer steuert nach dem Einsammeln der Dokumente die Aufbereitung der Daten. Nach der Datenaufbereitung werden die Daten mittels einer geeigneten Speicherkomponente persistent abgelegt. Der Zugriff auf die Daten wird über eine Anfragesprache, die verschiedene Speicherkonzepte bedienen kann, erfolgen. Die Problematik dieser Arbeit ist im Bereich des Gathering und der Datenaufbereitung einzuordnen. In der Abbildung 1.2 ist der momentane Stand des Gathering dargestellt.

Wie in [Bru00] beschrieben, ist die Grundlage dieser momentanen Realisierung die Architektur der Harvest–Suchmaschine. Harvest, von [BYRN99] als eine der wichtigsten Suchmaschinen im Internet[5] mit sehr vielen verschiedenen Installationen bezeichnet, bietet entscheidende Vorteile bei einer verteilten Architektur. Die Nachteile von Harvest sind durch fehlende Unterstützung neuerer

[3] Abstract — in diesem Kontext ein Begriff aus der Terminologie der Computerlinguistik.

[4] Eine weitere Nutzergruppe stellen die Betreiber der Suchmaschine dar. Ihre Schnittstelle ist im wesentlichen über die KA-Tools definiert.

[5] Zitat aus [BYRN99]: "Currently, there are hundreds of Harvest applications on the Web (for example, the CIA, the NASA, the US National Academy of Science, and the US Government Printing Office), as this software is on the public domain."

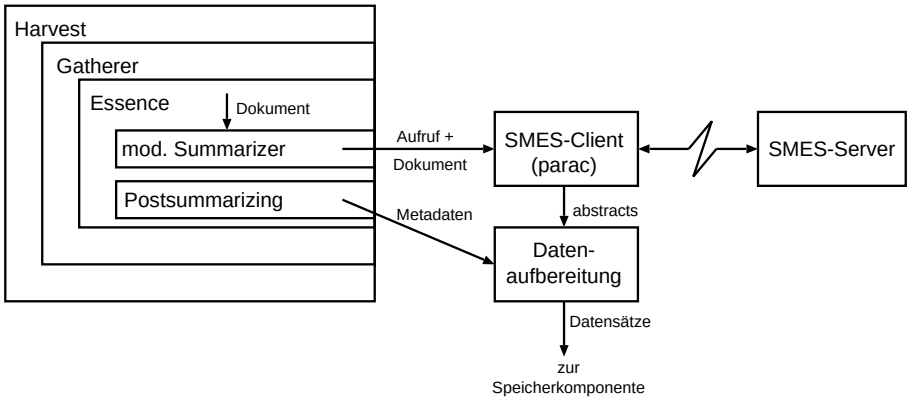

Abbildung 1.2: Gathering-Prozeß mit anschließender Datenaufbereitung und Speicherung

Dokumentformate, durch Einschränkungen beim internen Austauschformat und durch nicht unterstützte neuere Information Retrieval–Techniken (kurz IR–Techniken) gekennzeichnet.
Für GETESS ist bei Harvest insbesondere der Gatherer interessant. Der Gatherer liefert die zu sammelnden Dokumente in einer durch IR–Techniken zusammengefaßten Form. In der Abbildung 1.2 ist diese IR–Komponente als Summarizer dargestellt. An dieser Stelle wird ein natürlichsprachliches, wissensbasiertes Tool zur Syntax- und Semantikanalyse (SMES) eingebunden. Dieses Tool liefert ein sogenanntes abstract, welches eine Zusammenfassung in Form von Attributwertpaaren des Dokumentes enthält. Die abstracts und Metadaten aus dem Gathering–Prozeß bilden die Datengrundlage zur Datenaufbereitung und werden als Datensätze der Speicherkomponente zugeführt.
In diesem Szenario ist die natürlichsprachliche, wissensbasierte Analyse die einzige verwendete Möglichkeit der Wissensextraktion (die zusätzlichen Metadaten beschreiben nur inhaltsferne Dokumentattribute). Dieses Konzept der Inhaltserschließung ist für die im folgenden beschriebenen Problemstellungen allerdings nicht ausreichend. In Kapitel 3 wird die natürlichsprachliche, wissensbasierte Analyse mit SMES mit ihren Vor- und Nachteilen genauer untersucht.

1.3.2 Motivierende Problemstellungen und Ziele

Als Hauptproblempunkt wird sich die Informationsextraktion herausstellen. Dabei sind Konzepte und deren Integration zur Unterstützung der Informationsextraktion gefragt. Durch mehrere Extraktionskonzepte erhofft man sich eine bessere Informationsauslese der Dokumente und eine höhere Automatisierung (viele Realisierungen sind halbautomatisch) des Prozesses.
Das Ziel dieser Arbeit innerhalb von GETESS ist es, den bisherigen Gathering–Prozeß so zu erweitern, daß die bisherigen Gathering–Ergebnisse qualitativ verbessert werden. Dazu gibt es einige konkrete, teilweise weiterführende Problemstellungen. Die folgenden Problemstellungen werden aus dem GETESS–Szenario einer Suchmaschine über Internetdokumente motiviert.

- **Klassifizierung der Dokumente**
  Klassifizierte Dokumente erhöhen zum einen den Komfort für den Nutzer nur bestimmte Dokumente eines Themas auszuwählen (z.B. Übersichten oder spezielle Daten). Zum anderen ergeben sich zusätzliche Informationen zur Inhaltsbestimmung der Dokumente.
  Beispielsweise kann man Seiten klassifizieren, die Hotelzimmer beschreiben, im Gegensatz zu Seiten, die Listen von Hotels führen. Dabei kann die Klassifizierung syntaktisch oder semantisch bzw. strukturell oder inhaltlich erfolgen.

- **Erkennung ähnlicher Dokumente**
  Diese Problematik beinhaltet das Abgleichen von ähnlicher Information zwischen verschiedenen Dokumenten. Die Dokumente können dieselben Informationen haben oder sich ergänzen.

In diese Problemkategorie fällt ebenfalls die Erkennung redundanter Informationen. Man möchte Informationen möglichst nicht mehrfach speichern.
Ein Beispiel dazu wären zwei Hotelinformationsanbieter, die teilweise dieselben Hotels und teilweise Hotels exklusiv anbieten. Sie besitzen somit redundante Informationen und ergänzende Daten.
Das Erkennen solcher Ähnlichkeiten ist stark inhaltsbezogen. Je detaillierter und genauer man den Inhalt extrahiert, desto besser ist die Wirkungsweise der Konzepte zur Kombination der Daten.

- **Kombination verteilter Information**
  Ein weiteres Problem stellt sich bei der Annahme, daß zusammengehörige Inhalte auf mehrere Dokumente verteilt wurden und Beziehungen untereinander durch Links ausgedrückt werden. Ein Informationsanbieter z.B. stellt eine Hotelinformation durch eine Eingangsseite, mehrere Zimmerinformationsseiten plus der jeweiligen Preisseite dar. In der Datenbasis ist es günstiger diese Daten nicht getrennt pro Dokument, sondern zusammen als ein Datensatz mit mehreren zugehörigen Dokumenten zu betrachten.

Beispiel der angestrebten Informationsextraktion Im Anhang A ist eine Hotelseite beschrieben. Eine ideale Informationsextraktion dieses Dokuments könnte folgende Informationen ermitteln:

- Dies ist eine Hotelanbieterseite.
- Der Hotelname lautet Atrium Hotel Krüger.
- Die Adresse lautet 18069 Rostock usw.
- Die Zimmer haben Ausstattungen wie TV, Radio usw.
- Das Einzelzimmer kostet 95,00 - 125,00 DM mit Frühstück.
- Das Doppelzimmer kostet 150,00 DM mit Frühstück.
- Weitere Informationen wie Zahlungsmodalitäten, Besonderheiten.

Die Daten sind hier textuell aufgeführt. Ideal sind getypte Attributwertpaare, wie z.B. Hotelname = “Atrium Hotel Krüger” vom Typ String.
Eine solche Informationsextraktion läßt ein keyword–Index auf bisherigen Information Retrieval–Ansätzen nicht zu. Aktuelle Information Retrieval–Techniken ([Los98] und [BYRN99]) können mit keyword–Extraktion die Semantik von Web–Seiten nur sehr eingeschränkt erkennen.

Der gegenwärtige Stand der Forschung läßt keine hundertprozentige Inhaltserschließung zu. Eine allgemeine Lösung aller Problemstellungen in GETESS ist damit nicht zu erreichen. Eine Kombination aus neuen Konzepten verschiedener Ansätze und eine anwendungsorientierte Analyse sind besonders vielversprechend.

Ziel der Arbeit Mittels struktureller, linguistischer und domänenspezifischer Analysetechniken wird eine Informationsextraktion mit den oben genannten Eigenschaften konzipiert. Eine Klassifizierung von Dokumenten wird aufgrund von domänenspezifischen und anwendungsspezifischen Daten vorgenommen. Das Resultat der Informationsextraktion wird als abstract–Darstellung pro Informationspaket, welches auch über Dokumentgrenzen hinaus verteilt vorliegen kann, ausgedrückt. Es wird versucht, Daten, die gegenüber der Anwendung bzw. Anwendungsdomäne irrelevant sind, und Daten, die die Informationsextraktion verfälschen können, aus dem Ergebnis zu eliminieren.

1.4 Inhalt

Nach dieser Einführung und Motivation innerhalb der Internetsuchmaschine GETESS werden die Konzepte zur Unterstützung der Informationsextraktion betrachtet. Dazu gehören Daten, die aus den Dokumenten oder ihrer Umgebung extrahiert oder analysiert werden. Die strukturelle, domänenspezifische und linguistische Analyse stellt hier eine komplexe Erweiterung bestehener Analyseschritte in herkömmlichen Suchmaschinen dar. Die domänenspezifische, linguistische Analyse wird in Kapitel 3 explizit vorgestellt.
In Kapitel 4 werden nach Betrachtungen zu einigen bisherigen Forschungsansätzen die eigenen konzeptionellen Lösungsansätze der in Kapitel 1 motivierten Problemstellungen dargelegt. Dazu gehört eine konzeptionelle Architektur, eine Beschreibung der einzelnen Komponenten und die resultierenden Datenstrukturen.
Kapitel 5 zeigt dann den Vorschlag für eine Umsetzung innerhalb der GETESS–Gatherer–Umgebung, wobei die Umsetzung der einzelnen Komponenten an einer konkreten GETESS–Gatherer–Architektur beschrieben werden. Anschließend werden Beispiele gezeigt und die dargestellten Ansätze bewertet und evaluiert.
Die Arbeit wird durch eine Zusammenfassung der Arbeit und einen umfangreichen Ausblick auf weitere interessante Fragestellungen abgeschlossen.

Kapitel 2

Konzepte zur Unterstützung der Informationsextraktion

In diesem Kapitel werden Konzepte betrachtet, die einerseits elementar zu den Informationen eines Dokuments führen und andererseits zusätzlich, als Metadaten oder externe Daten, die Informationsextraktion unterstützen.
Die möglichen Daten werden in diesem Abschnitt zuerst klassifiziert. Aus den klassifizierten Daten werden jene ausgewählt, die die Informationsextraktion in sinnvoller Weise unterstützen können. Zu den ausgewählten Konzepten werden dann mögliche Modellierungen diskutiert.

2.1 Klassifikation möglicher Daten zur Unterstützung

Nachfolgend werden Informationen, die beim Gathering–Prozeß zur Verfügung stehen können, aufgezählt und näher erläutert. Außerdem werden Herkunft und Nutzen der einzelnen Daten diskutiert.

Die vorgenommene Klassifizierung bezieht sich auf die Herkunft der Daten. Es ist möglich, auch andere Klassifizierungen vorzunehmen, beispielsweise nach Art der Daten, Größe oder Aufwand. Eine Klassifizierung nach Nutzen oder durch Gewichtung der Verwendung ist nur mit dem speziellen Anwendungsfall zu vereinbaren.

2.1.1 Dokumentübergreifende Daten

Die dokumentübergreifenden Daten werden in diesem Szenario vor allem durch die Struktur des Internet ausgedrückt. Die Struktur des Internet baut sich in erster Linie durch Links auf. Eine Verfolgung der Links gibt Aufschluß über Beziehungen zwischen Dokumenten und Dokumentgruppen.

- **Linkstruktur zwischen Domänen**
  Die Domäne wird durch einen Informationsanbieter im Internet aufgespannt. Dazu zählen alle Seiten, die der Informationsanbieter innerhalb der Internet–Hauptadresse (domain, wie z.B. “uni-rostock.de”) angeordnet hat.
  Beziehungen zwischen solchen Domänen geben Informationen über die nähere Umgebung eines durchsuchten Informationsanbieters im Internet (z.B. kann ein Hotelinformationsanbieter neben eigenen Seiten zu Hotels auch Links auf Seiten von den Hotels selber anbieten). Solche Informationen werden im allgemeinen nicht von den Gatherern ausgewertet.

- **Linkstruktur der Domäne:**
  Die Struktur der Seiten kann Informationen über Verteilung, Anordnung und Beziehungen der Seiteninformationen enthalten.
  Ein Gatherer kann diese Struktur liefern, da er intern diese Daten zur Gewinnung von Dokumenten nutzt.

2.1.2 Dokumentdaten

- **Allgemeine Dokumentmetadaten**
  Mit Metadaten sind hier Daten gemeint, die das Dokument selbst beschreiben. Das können Daten sein, die durch den Informationsanbieter mitgeliefert werden oder Daten, die beim Gathering zum Dokument festgestellt wurden. Die folgenden Punkte stellen eine kleine Auswahl solcher Daten vor.

  - **Dokumenttyp**
    Der Dokumenttyp gibt an, wie das Dokument behandelt werden muß, beispielsweise welches Informationsextraktionsprogramm anzuwenden ist. Der Dokumenttyp wird von den meisten Suchmaschinen ermittelt und diese Suchmaschinen haben im allgemeinen dokumenttypabhängige Funktionalitäten.
  - **Größe**
    Die Größe einer Datei kann Informationen über Kosten und Aufwand der Verarbeitung des Dokumentes geben. Manche Suchmaschinen, wie z.B. Harvest nutzen diese Information auch zur Anfrage.
  - **Prüfsumme über ein Dokument**
    Eine Prüfsumme über ein Dokument ist vor allem sinnvoll, wenn man Operationen auf den Dokumenten optimieren möchte. Beispielsweise beim Vergleich zweier Dokumente genügt ein Vergleich der Prüfsummen. Viele Suchmaschinen benutzen diesen Mechanismus zum Abgleich der Originaldaten zu den Daten in der Datenbasis der Suchmaschine (Harvest benutzt MD5–Prüfsummen).
  - **Zeitabhängige Daten**
    Für jedes Dokument ist es sinnvoll, zeitabhängige Daten, wie Zeitpunkt des Einsammelns, Zeitpunkt des nächsten Updates oder maximale Lebenszeit[1] der Informationen bei fehlschlagendem Update, zu halten. Diese Daten sind einmal für das Sammeln bzw. Aktualisieren nützlich, zum anderen werden die Daten bei der Speicherung benötigt.

- **Dokumenttypabhängige Daten**
  Diese Daten sind durch die interne Struktur der Dokumente gekennzeichnet. Sie sind abhängig vom Dokumenttyp. Die folgenden Dateitypen sind als Beispiele für diese Art der Daten zu sehen.

  - **HTML**
    HTML (Hypertext Markup Language) als momentaner Dokumentstandard im Internet hat Strukturelemente, Tags genannt, die wichtige Informationen über den Inhalt eines Dokumentes enthalten. Aufbau und Inhalt von HTML sind unter anderem in [Tol97] zu finden.
  - **XML**
    XML (eXtensible Markup Language) [BM98] als Vorschlag des W3C (World Wide Web–Consortium) für einen neuen, erweiterten Dokumentstandard im Internet bietet eine explizite Strukturdefinition (Document Type Definition, DTD) und damit eine verbesserte Inhaltsanalyse. Bis sich dieser Standard allerdings auf breiter Anwendungsebene durchsetzt, ist abzuwarten.
  - **JPG, GIF**
    Viele Informationen werden in Bilddateien untergebracht, beispielsweise Überschriften, Links oder Navigationselemente (Map-Dateien). Um solche Informationen zu extrahieren, ist es nötig, geeignete Bildanalyseprogramme einzusetzen. Hier stellt sich dann die Frage nach dem Verhältnis zwischen Aufwand und Nutzen, da aktuelle Bildanalyseprogramme nicht zufriedenstellend arbeiten.

[1] Im allgemeinen sind Internetdaten recht kurzlebig bzw. erfahren häufige Updates oder sind schnell veraltet.

  - **Programmbestandteile**
    Programme oder Programmbestandteile, wie Java–Applets, Javascript oder Jscript, werden bei herkömmlichen Suchmaschinen nicht ausgewertet. Inwieweit solche Daten für die Inhaltsauswertung interessant sind, entscheidet meist der konkrete Anwendungsfall. Eine Programmanalyse während des Gathering ist als sehr aufwendig anzusehen.

- **Dokumentinhalte**
  Über die Erschließung der Dokumentinhalte erhält man die eigentliche Dokumentinformation. Diese Daten können als Menge kleiner "Informationsbrocken" oder auch als eine komplett semantisch aufbereitete Bedeutungsinformation vorliegen.

  - **Keyword–Extraktion**
    Die aus dem Gebiet des Information Retrieval bekannten Indexierungs- und Zugriffsverfahren [Fuh97], wie z.B. invertierte Listen, sind für die einfache Inhaltserschließung geeignet. Dazu werden durch eine keyword–Extraktion Identifikatoren der Dokumente gebildet. Diese Identifikatoren können dann semantisch weiterbearbeitet werden, um eine relativ allgemeine Bedeutung des Inhalts zu bestimmen.
  - **Linguistische Analyse**
    Unter Anwendung natürlichsprachlicher Parsing–Programme erhält man linguistische Informationen, wie z.B. morphologische Daten oder Daten über Satz- und Textaufbau. Durch die Interpretation solcher Daten können semantische Beziehungen innerhalb des Textes erkannt werden.
  - **Semantische Informationen**
    Innerhalb des NL–Parsing ist es möglich, semantische Informationen zu extrahieren. Eine semantische Analyse kann nach [GW93] auf verschiedenen Wegen durchgeführt werden.

2.1.3 Externe Wissensquellen

- **Wissensrepräsentationen**
  Wissensrepräsentationen liefern Konzepte bzw. Konzeptstrukturen. In Verbindung mit der semantischen Analyse beim NL–Parsing kann die Semantik eines Dokumentes bezüglich des Wissens geeigneter Wissensrepräsentationen bestimmt werden.
  Interessante, mögliche Wissensquellen sind Ontologien oder Thesauri.

- **Datenbanktypische Speicherkomponente**
  Benutzt man für die gesammelten Daten eine Datenbank, können auf der Datenbank Integrationsregeln definiert werden. Diese Regeln können Ergebnisse hervorrufen, die für die Informationsextraktion von Bedeutung sind. Beispielsweise könnte eine Primärschlüsselverletzung auf teilweise redundante Informationen hindeuten.

2.1.4 Zusammenfassung

Die Klassenhierarchie ist in der Abbildung 2.1 einmal übersichtlich dargestellt.

2.2 Auswahl relevanter Konzepte

In diesem Abschnitt werden für den GETESS–Kontext relevante Konzepte ausgewählt und ihre Modellierung diskutiert. Weiterhin werden die Konzepte GETESS-bezogen konkretisiert.

2.2.1 Auswahlkriterien

Die Auswahl wird nach folgenden Gesichtspunkten stattfinden:

- **Nutzen**
  Der Nutzen beschreibt die mögliche Verbesserung des Extraktionsergebnisses bei Integration der jeweiligen Informationen.

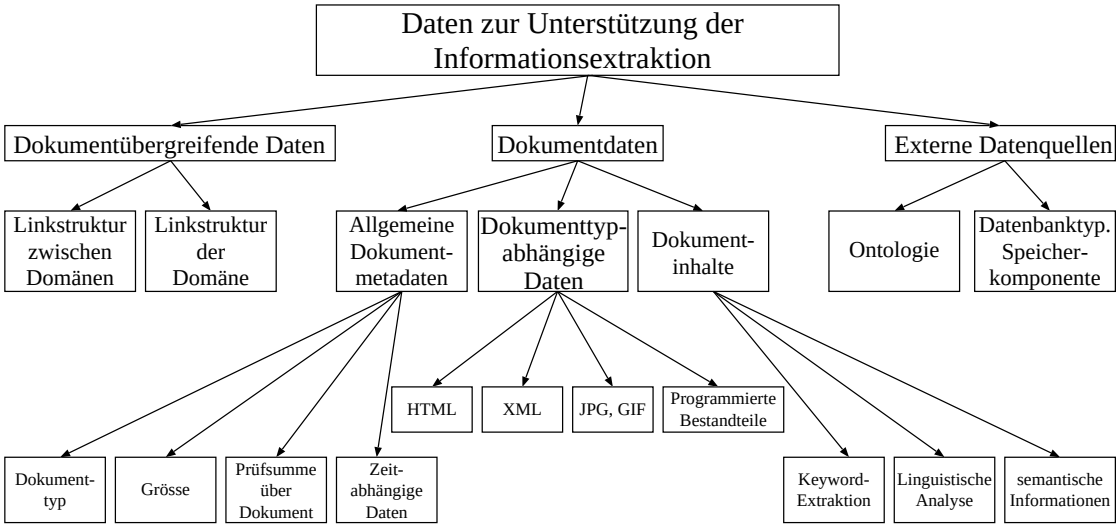

Abbildung 2.1: Übersicht der Klassifizierung der Daten zur Informationsextraktion

- **Aufwand**
  Darunter ist der konzeptionelle Aufwand und der Implementierungsaufwand beim Bereitstellen der Daten zu verstehen.
  Der Aufwand ist im Verhältnis zum Nutzen zu betrachten. Ein geringer Nutzen bei sehr hohem Aufwand ist im GETESS-Kontext nicht akzeptabel.

- **Zeitkomplexität**
  Die in [Bru00] erkannte Zeitproblematik beim Gathering–Prozeß im GETESS–Projekt erfordert in dieser Hinsicht eine Bewertung der Konzepte. Dabei kann die Zeitkomplexität bei einigen Konzepten nicht genau bzw. nur vage festgelegt werden.

2.2.2 Betrachtung der Daten nach Kriterien

Im folgenden werden die Daten nach den erwähnten Kriterien untersucht.

Dokumentübergreifende Daten Die Linkstrukturen (sowohl zwischen Domänen als auch innerhalb einer Domäne) werden zum Einsammeln der Dokumente benötigt, da die Sammelagenten durch die Linkstrukturen navigieren. Der Aufwand, diese Daten für die Informationsextraktion zu sammeln, ist deshalb gering. Der Nutzen läßt sich durch die Motivation der verteilten Informationen über mehrere Dokumente annehmen. Diese Linkstrukturen führen auf weitere potentielle, dem Informationspaket (eine komplexe Informationseinheit, wie z.B. ein Hotel mit Zimmern, Zimmerpreisen, Anschrift und Ausstattung) angehörige Informationen. Die Zeitkomplexität beim Beschaffen dieser Informationen ist vernachlässigbar, da die Informationen für das Sammeln ohnehin benötigt und bereitgestellt werden. Die Verarbeitung dieser Daten kann mit der Zeitkomplexität von Graphennavigationstechniken gleichgestellt werden, da die Link–Dokument–Struktur mittels Graphen darstellbar ist.

Allgemeine Dokumentmetadaten Die Dokumentmetadaten werden beim Gathering gebildet und liegen deshalb größtenteils vor. Sie werden für die richtige Bearbeitung der Dokumente und zum geregelten Ablauf (Update, Delete von gesammelten Daten) des Gathering benötigt. Ein direkter Nutzen für die Informationsextraktion ist für diese Daten nicht zu sehen (mit Ausnahme des Dokumenttyps, der im wesentlichen die richtige Verarbeitungsstrategie vorgibt). Allerdings können diese Angaben dem Nutzer als zusätzliche Daten über das Dokument bereitgestellt werden

und ihm weitere Auswahlkriterien bieten. Die Daten liegen vor, und eine direkte Verarbeitung ist nicht vorgesehen, somit wird keine zusätzliche Zeitkapazität benötigt.

Dokumenttypabhängige Daten Allgemein ist es wünschenswert, möglichst viele Dokumenttypen unterstützen zu können. Aufwand und Nutzen beim Verarbeiten der einzelnen Internet–Dokumenttypen variieren allerdings sehr stark, und es ist sinnvoll, einige wenige zu selektieren.
Da die meisten Informationen momentan im HTML–Format vorliegen, wird das Bearbeiten dieses Formats in dieser Arbeit vorrangig betrachtet. Zum Parsen solcher HTML–Dateien existieren umfangreiche Bibliotheken mit recht schnellen Algorithmen in fast jeder Programmiersprache. Dabei entstehen meist baumförmige Strukturen, in denen navigiert werden kann. Teilweise parst der Gatherer die Dateien (z.B. zur Extraktion der Links).
Das Format XML, welches HTML nach Wunsch des W3C als Internet–Dokumentsprache ersetzen soll, wird in dieser Arbeit nur ansatzweise betrachtet. XML–Dokumente als Hypertext–Seiten im Internet sind derzeit kaum zu finden.
Bilddaten, wie GIF oder JPG, können wichtige Daten enthalten. Eine Analyse solcher Daten ist momentan mit sehr großem Aufwand verbunden und im allgemeinen qualitativ nicht befriedigend. Aus diesem Grunde werden Bilddaten in dieser Arbeit nicht weiter betrachtet.
Ähnlich verhält es sich mit Programmdaten, wie z.B. Java Applets, Javascript und viele andere, teilweise proprietäre, Internet–fähige Programme. Auch hier stehen Aufwand und Nutzen im GETESS–Kontext in einem schlechten Verhältnis.

Dokumentinhalte Die linguistische Analyse mit anschließender semantischer Analyse der Internet–Dokumente spielt im GETESS–Kontext eine wesentliche Rolle. Der konzeptionelle und implementationstechnische Aufwand ist sehr groß. Die daraus resultierenden, typisierten und semantisch erschlossenen abstracts der Dokumente ergeben dagegen einen sehr großen Nutzen. Die linguistische und semantische Analyse der Texte durch SMES/Ontologie wird das Kernstück der Inhaltserschließung bei GETESS darstellen und in Kapitel 3 genau beschrieben. In Kapitel 3 wird auch die Zeitkomplexität beschrieben.
Da der bei GETESS zugrundeliegende Gatherer eine keyword–Extraktion vornimmt, ist es relativ einfach, diese keyword–Liste bei der weiteren Inhaltserschließung einzusetzen. Einerseits wäre die Auswertung der keyword–Listen interessant für die Evaluierung und Qualitätseinschätzung des Systems und andererseits könnte eine Integration der keywords und der linguistisch–semantischen Analyse das Ergebnis verbessern.

Externe Datenquellen Wissensrepräsentationen, wie Ontologien oder Thesauri, benötigen zum Aufbau einen relativ großen Aufwand. Die semantische Analyse verwendet im allgemeinen aufbereitetes Wissen. In GETESS wird dafür eine Ontologie eingesetzt. Der Einsatz bei der Textanalyse wird in Kapitel 3 gezeigt. Die Zeitkomplexität beim Zugriff auf die Ontologie ist auf die Navigation in semantischen Netzen zurückzuführen.
Statistische Daten der Speicherkomponente lassen sich relativ schnell mittels datenbankspezifischer Operationen generieren. Ungünstige zeitliche Einflüsse solcher Operationen auf die Nutzersuchzugriffe sollten vermieden werden. Der mögliche Nutzen dieser Daten ist momentan sehr gering. Daher wird die Kopplung dieser Daten an die Informationsextraktion in dieser Arbeit nur am Rande betrachtet.

2.2.3 Tabellarische Zusammenfassung

Die folgende Tabelle 2.1 faßt die hier erläuterte Auswahl der verschiedenen Konzepte zur Unterstützung der Informationsextraktion zusammen.

2.3 Modellierung relevanter Konzepte

Die Frage der Modellierung ist im Zusammenhang mit Kapitel 4, den Lösungsansätzen zur Informationsextraktion, zu sehen. In Kapitel 4 werden die hier betrachteten Konzepte zu einer sinnvol-

Konzept	*Nutzen*	*Aufwand*	*Zeitkomplexität*	*Auswahl*
Linkstrukturen	+	+	+	ja
Metadaten	-	+	+	ja, zusätzliche Daten
Dokumenttyp–Daten				
HTML	+	/	/	ja
XML	+	/	/	Zukunft
Bilddaten	/	-	-	nein
Programmdaten	/	-	-	nein
Dokumentinhalte				
Keywords	+	/	/	ja
Linguistik/Semantik	+	-	-	ja
Ontologie	+	-	/	ja
Datenbankdaten	/	+	+	nein

Tabelle 2.1: Zusammenfassende Auswahl relevanter Konzepte (+ – positiv, / – normal, - – negativ).

len Informationsextraktion kombiniert. Dazu ist es hilfreich, wenn die Modellierung der einzelnen Konzepte im wesentlichen syntaktisch und semantisch untereinander abgestimmt ist.

2.3.1 Modellierung dokumentübergreifender Daten

Die dokumentübergreifenden Daten sind in diesem Szenario Internet–Linkstrukturen. In [FLM98] wird das WWW durch ein Graphenmodell und durch ein semistrukturiertes Datenmodell (semistrukturierte Daten sind Daten mit teilweise erkennbarem Schema) beschrieben. Zur Modellierung der Seiten–Link–Strukturen im WWW sind Graphen besonders gut geeignet. In [Bru99] sind einige Ansätze zur Graphenmodellierung betrachtet worden.
Strudel [FFLS97] und Lore [AQM+96], zwei Webseiten–Management Systeme, nutzen Graphenmodelle zur Repräsentation der Daten. Beide Modelle beruhen im wesentlichen auf dem OEM (Object Exchange Model) [PGMW95]. In diesem semistrukturierten Datenmodell gibt es einfache und komplexe Objekte. Alle Objekte sind durch einen Object Identifier (OID) eindeutig gekennzeichnet. Die Objekte haben weiterhin einen Namen (Label), einen Typ und einen Wert. In einem zugehörigen Graphen stellen die Objekte die Knoten dar. Die einfachen Knoten bilden die Blätter des Graphen, haben einen einfachen Datentyp und keine ausgehenden Kanten. Die komplexen Knoten sind vom Typ Menge und haben mindestens eine ausgehende Kante. Die Kanten bilden die Beziehungen zwischen den Objekten und sind durch Label beschriftet. Durch die Label–Namen sind die jeweiligen Objekte erreichbar. Ein Beispiel für einen OEM–Graphen ist in Abbildung 2.2 zu sehen.

Eine Abbildung der Dokument–Link–Strukturen auf die OEM–Graphen kann auf einfache Weise durchgeführt werden. Die Dokumente sind die Objekte und bilden die Knoten, wobei Dokumente, die keine Links enthalten, im OEM einfache Knoten darstellen. Die Links stellen die Kanten dar. Die Kanten–Label werden entweder durch den kompletten Link-Pfadnamen oder durch den Namen des zu erreichenden Objektes belegt. Daraus ergeben sich für WWW–Ausschnitte recht große, grafische Netze.

2.3.2 Modellierung Dokumentdaten

Ein Internet–Dokument vom Typ HTML oder XML ist nach einer baumförmigen Struktur aufgebaut. Auch hier eignen sich Graphenmodelle, wie z.B. OEM–Graphen. Dies bietet sich insofern an, da externe Linkstrukturen auf die gleiche Art und Weise dargestellt werden können (siehe oben). In diesem Falle begrenzen die strukturellen Bezeichner (z.B. HTML–Tags) die Objekte und legen den Label fest. Die einfachen Objekte, die Blätter im Graphen, werden die Texte des Dokumentes enthalten. Alle anderen Objekte sind komplex und haben eine Menge von weiteren Objekten. Über

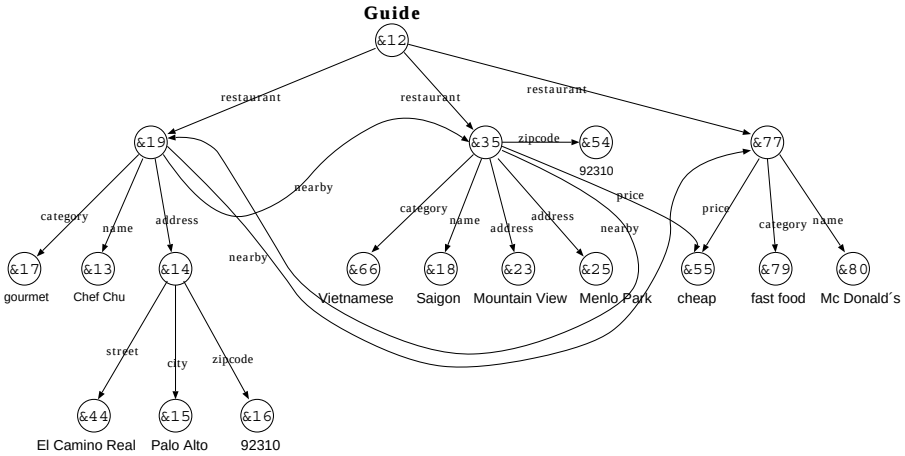

Abbildung 2.2: Beispiel eines OEM–Graphen nach [AQM+96]

diese Mengen wird sich die Baumhierarchie des Dokuments aufbauen.

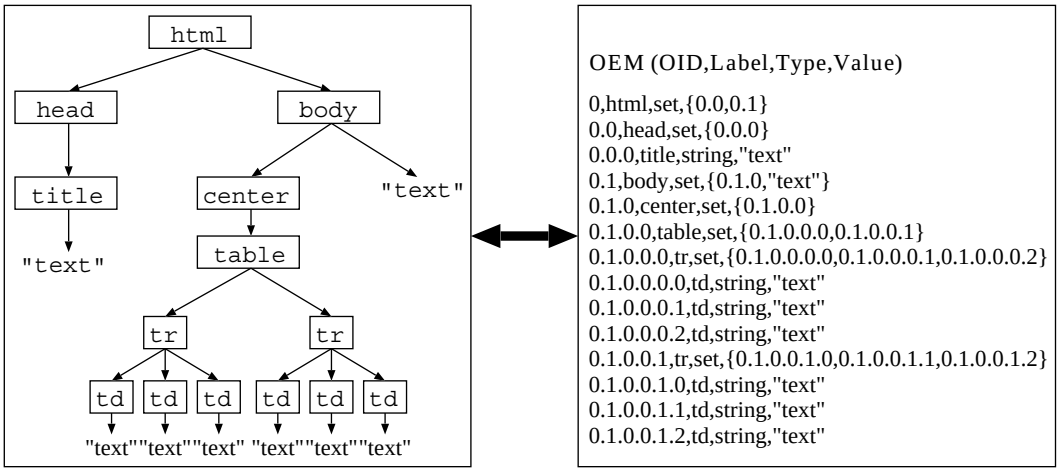

Abbildung 2.3: Abbildung eines HTML–Baums auf OEM

Abbildung 2.3 zeigt einen solchen HTML–Baum und die dazugehörige OEM–Abbildung. Die OEM–Abbildung wird in Form eines 4–Tupels (eine Möglichkeit der Implementation), und zwar Label, Typ, Wert und OID, dargestellt, wobei die OIDs die Hierarchie widerspiegeln.
XML–Dokumente (XML–Modell siehe [BM98]) sind durch ihre konsequente Trennung der Struktur und den Daten von der Darstellungsweise besser zu modellieren als HTML–Dokumente. Dabei ist es nicht einmal nötig die XML–Daten in eine andere Modellierung zu überführen. Es bietet sich im Gegenteil an, die Umgebung ebenfalls in XML zu modellieren oder zumindest die XML–Darstellung in die Umgebung zu integrieren.

2.3.3 Modellierung Dokumentinhalte

Daten aus der keyword–Extraktion sind Eigenschaften bzw. Attribute des jeweiligen Dokuments oder des Dokumentausschnittes. Die Modellierung dieser Daten geschieht demnach auf Basis der jeweiligen Dokumentmodellierung. Ebendso können weitere mögliche, relevante Dokumentattribute

(Dokumentmetadaten) eingebaut werden, wie z.B. Dokumenttyp oder Updatetime bzw. Deletetime.
Die linguistischen, semantischen Daten werden bei GETESS mittels natürlichsprachlicher Parser und Ontologien gewonnen. Darauf wird im Kapitel 3 genauer eingegangen und auch die Frage der Modellierung geklärt.

2.3.4 Externe Datenquellen, Ontologie

Als externe Datenquellen werden hier Ontologien betrachtet. Die Ontologie ist ein semantisches Netz über die Konzepte einer speziellen Wissensdomäne. Eine Anfrage zu den Konzepten und ihren Beziehungen geschieht über eine Inferenzmaschine. Beispielsweise kann man nach einer möglichen Beziehung zwischen zwei angegebenen Konzepten anfragen. Man erhält darauf ein Tripel mit den zwei Konzepten und deren Beziehung. Diese Konzepte (durch domänenspezifische keywords motiviert) und die Beziehung sind Eigenschaften des jeweiligen Dokuments und werden dementsprechend innerhalb dessen Dokumenteigenschaften modelliert. Werden durch die Ontologie–Anfrage zwei Konzepte unterschiedlicher Dokumente (Kombination verteilter Information) in Beziehung gebracht, sind diese Beziehungen an die Verbindung (z.B. Links) zwischen den Dokumenten als Attribut des Links gebunden. Die genaue Implementierung wird diese Ansätze vertiefen.
Der konkrete Aufbau und die Funktionsweise der Ontologie in GETESS wird in Kapitel 3 im Zusammenhang mit der natürlichsprachlichen Analyse vorgestellt.

2.3.5 Zusammenfassung

Die Modellierung der Konzepte stützt sich vor allem auf eine graphische Darstellung der Internet–Strukturen. Die Konzepte sind meist semistrukturiert und können mittels OEM einfach dargestellt werden. Weitere Aspekte, wie Eigenschaften, Attribute und Beziehungen zwischen Eigenschaften, werden in die OEM–Darstellung integriert bzw. erweitern das OEM.
Ein weiterer Aspekt ist, daß unterschiedliche Modelle auf verschiedenen Ebenen der Verarbeitung existieren. Hier wurden die Daten unmittelbar nach dem Sammeln bis zur Verarbeitung betrachtet. Nach der Verarbeitung werden die Daten an die Schnittstelle zur Speicherkomponente genormt übergeben. Diese Schnittstelle wird durch eine XML–Darstellung festgelegt.

Die Modellierung der Datenkonzepte ist der zweite Schritt zur kombinierten Informationsextraktion in GETESS nach der ersten Betrachtung und Auswahl der möglichen Konzepte. Der nächste Schritt zur konkreten Darstellung der Daten wird in Kapitel 4 erfolgen. In Kapitel 4 werden Möglichkeiten zur Kombination der Daten zu einer möglichst ergiebigen Informationsextraktion betrachtet. Die Frage der Kombination wird über die konkrete Modellierung entscheiden. Die Modellierung ist weiterhin abhängig von der jeweiligen Implementierung und der genauen Umgebung. Die Implementierung in Kapitel 5 wird dann die endgültige Modellierung der Konzepte in GETESS darlegen.

Kapitel 3

NL–Parsing in GETESS

Das natürlichsprachliche Parsing der Dokumente in GETESS hat die Zielstellung, ein möglichst genaues, semantisch erschlossenes Abbild des Dokumentes in Form eines abstract zu generieren. Dieses abstract, als Inhaltsrepräsentator fungierend, wird von computerlinguistischer Software (SMES [NM98]) vom DFKI Saarbrücken und durch wissensbasierte Techniken (Ontologie [FDES97]) von der Universität Karlsruhe aufgebaut. Diese beiden Komponenten werden in GETESS integriert und nachfolgend als SMES/Ontologie bezeichnet.

Zunächst werden einige Grundlagen dieser Wissensgebiete dargelegt. Anschließend werden die Komponenten vorgestellt und die Integration diskutiert. Zum Ende des Kapitels wird die momentane Umsetzung getestet und bewertet.

3.1 Grundlagen des NL–Parsings in GETESS

Bevor das NL–Parsing mittels SMES/Ontologie vorgestellt wird, führt dieser Abschnitt in die allgemeinen Begriffe und Konzepte der beiden Komponenten, SMES und Ontologie, ein.

3.1.1 Terminologie und Konzepte der natürlichsprachlichen, linguistischen Analyse

An dieser Stelle werden wichtige Begriffe der Computerlinguistik und wichtige Konzepte des natürlichsprachlichen Parsing eingeführt und erläutert.

Begriffe Der Begriff des abstracts als wesentliches Konzept im GETESS–Kontext wird in diesem Zusammenhang nun genau definiert. Ein abstract ist eine Menge von instanziierten Templates bzw. domänenspezifischen Konzeptbeschreibungen. Für GETESS bedeutet dies, daß aus dem gegebenen Dokument eine Zusammenstellung von domänenspezifischen Informationen des jeweiligen Inhalts des Dokumentes generiert wird. Ein Template beschreibt in diesem Zusammenhang ein Konzept, welches, wenn es im Dokument vorkommt, mit den Werten aus dem Inhalt des Dokumentes instanziiert und in die Menge der abstract–Beschreibungen aufgenommen wird. Aufbau und Realisierungen dieser abstracts sind im nächsten Abschnitt beschrieben.

Nachfolgend werden einige Begriffe der Computerlinguistik nach [Bus90] kurz erläutert. Dies ist nur eine sehr kleine Auswahl von besonders relevanten Aspekten innerhalb dieser Arbeit.

- Morphologie — Oberbegriff für Flexion und Wortbildung in der Sprachwissenschaft.
- Grammatik — systematische Beschreibung der morphologischen und syntaktischen Regularitäten einer natürlichen Sprache.
- Lexikon — beschreibt den Wortschatz einer Sprache soweit kodifiziert, als seine Formen und Bedeutungen nicht aus den Regularitäten des Sprachsystems ableitbar sind.
- Syntaxbaum — formale Beschreibung der syntaktischen Analyse einer natürlichsprachlichen Äußerung.

- Phrase — aus dem Englischen übernommene Bezeichnung für syntaktisch zusammengehörige Wortgruppen.
- Nominalphrase (NP) — Phrase, die ein Nomen oder Pronomen als Kern enthält und in unterschiedlicher Weise erweitert sein kann.
- Präpositionalphrase (PP) — Phrase mit unterschiedlicher kategorialer Ausprägung, die vor allem die syntaktischen Funktionen des Adverbials, des Attributs oder des Objektes erfüllt.
- Verbalphrase (Verbkomplex) — Phrase, die als unmittelbare Konstituente des Satzes fungiert und obligatorisch ein Verb enthält.

Konzepte Bei natürlichsprachlichen Systemen unterscheidet [GW93] die theoretische und die Ingenieur–Perspektive. SMES ist ein System, das aus der theoretischen Sicht entwickelt wurde. Das heißt, das System beinhaltet vordergründig theoretische Fragestellungen zur Linguistik. Eine Anwendersicht, wie in der Ingenieur–Perspektive, wird beim konkreten Einsatz der SMES–Software bestimmt und als eigenständige Komponente umgesetzt.

Allgemein beinhaltet die Architektur der natürlichsprachlichen Textverarbeitung eine syntaktische und eine semantische Analyse.

In der **syntaktischen Analyse** werden Parser, Grammatiken und Lexika benötigt. Der Input in Form eines natürlichsprachlichen Textes wird mittels Parser unter Nutzung der Grammatiken und Lexika allgemein in Syntaxbäume aufgelöst. Bei der lexikalischen Analyse, als ersten Teil der Syntaxerkennung, werden mittels Lexika und unter Berücksichtigung der Morphologie die lexikalischen Einheiten separiert und linguistische Eigenschaften bestimmt. Die lexikalischen Einheiten mit ihren Eigenschaften werden nun durch die Regeln der Grammatik in einen Syntaxbaum oder mehrere Syntaxbäume transferiert. Der Syntaxbaum präsentiert eine logische Abstraktion aus den einzelnen Lexemen zu logisch zusammengehörigen Einheiten. In Abbildung 3.1 wird ein solcher Syntaxbaum an einem einfachen Beispielsatz gezeigt.

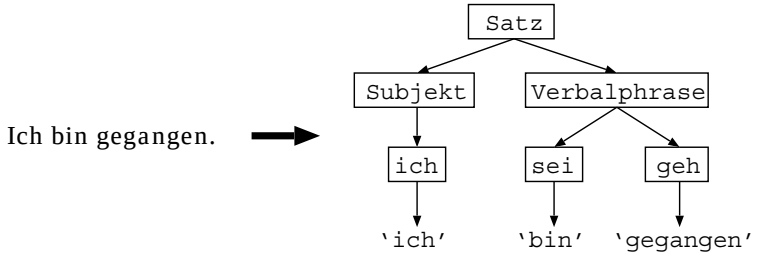

Abbildung 3.1: Beispiel eines Syntaxbaums für den Satz "Ich bin gegangen"

In der **semantischen Analyse** werden die syntaktisch gewonnenen Daten mit Bedeutungsinformationen der Daten angereichert. Die semantische Analyse ist meist eng mit der syntaktischen Analyse verbunden. Prinzipiell besteht die Möglichkeit, Syntax und Semantik getrennt, im direkten Zusammenhang oder als Einheit zu analysieren.
Beim natürlichsprachlichen Parsen wird von den Bedeutungen der Lexeme einer Phrase auf die Bedeutung der Phrase geschlossen und von den Phrasen auf die jeweiligen Sätze. Dies nennt man die Ermittlung der Satzsemantik. Dieses Prinzip läßt sich rekursiv auf Textabschnitte, Dokumente, Dokumentgruppen erweitern. Dabei wächst allerdings der Aufwand schnell und bei jedem Rekursionsschritt wird das Ergebnis ungenauer. Außerdem können wichtige Bedeutungsinhalte zwischen Einheiten verschiedener Ebenen bestehen, so daß eine Analyse über den vollen Umfang aussichtslos erscheint. Notwendige Einschränkungen sind beim zu suchenden Wissen und bei den Rekursionsebenen sinnvoll.

3.1.2 Ontologie

Eine Ontologie, als relationales Netz aus Konzepten, beschreibt ein ausgewähltes, möglichst abgeschlossenes Wissensgebiet. Das Wissen aus dem gewählten Bedeutungsraum kann angesprochen werden und Zusammenhänge können erfragt werden.
Für GETESS ist eine derartige Eingrenzung des Bedeutungsraums sehr wichtig, da damit ein geringer Aufwand und eine höhere Effektivität der natürlichsprachlichen Informationsextraktion verbunden ist.

In GETESS wird eine Ontologie als Modell eines Ausschnitts der allgemeinen Begriffswelt betrachtet. Die zu einer ausgewählten Domäne gehörenden Konzepte werden mit ihren Begriffen und den Beziehungen untereinander dargestellt. Weiterhin ist es möglich, Regeln bzw. Methoden auf den Begriffen zu definieren.

Neben der Modellierung und Darstellung der Konzepte ist der Zugriff auf und die Navigation in der Ontologie eine wesentliche Aufgabe. Beim Zugriff auf die Ontologie bei Anfragen, wie z.B. eine Beziehung zwischen Hotel und Zimmer finden, wird mittels einer Inferenzmaschine die benötigte Information aus den Ontologiedaten generiert. Weitere diesbezügliche Zusammenhänge sind in [FDES97] zu finden. Eine Visualisierung zur nutzergestützten Navigation kann z.B. mittels Hyperbolic Ontology View realisiert werden. [BPW+98] beschreibt die momentane Visualisierung der Ontologie in GETESS.

Eine solche Ontologie ist als semantisches Referenzmodell im GETESS–System geeignet die natürlichsprachliche Informationsextraktion sinnvoll zu unterstützen.

3.2 Aufbau und Funktionsweise des NL–Parsing in GETESS

Zunächst werden die Anforderungen und die wesentlichen Eigenschaften des NL–Parsing diskutiert. Auf Grundlage der darauf folgend beschriebenen Architektur wird die Funktionsweise der Software vorgestellt.

3.2.1 Aufgaben und Eigenschaften des NL–Parsing in GETESS

Aus dem Kapitel 1 ist bekannt, daß der Einsatz des NL–Parsing zum einen zur Auswertung der Nutzeranfragen im Dialogmodul und zum anderen zur Informationsextraktion aus den gesammelten Dokumenten beim Gathering–Prozeß benötigt wird.
Aus diesen Aufgaben heraus ergeben sich relevante Eigenschaften der natürlichsprachlichen Textverarbeitung bezüglich des Gathering–Prozesses und der Dialogkomponente.

GETESS–relevante Eigenschaften In [BPW+98] werden fünf Eigenschaften (die teilweise einander bedingen) genannt.

- **Robuste und effiziente Sprachverarbeitung** von deutschen Texten:
  Die robuste und effiziente Verarbeitung ist ein Ergebnis der sehr flachen Sprachverarbeitung und Informationsextraktion. Es wird innerhalb von GETESS in diesem Bezug eine große Herausforderung an die Geschwindigkeit bestehen. Die Problematik besteht dabei nicht nur bei der Extraktion von Informationen aus dem Internet, sondern auch beim natürlichsprachlichen Dialog mit dem Nutzer.

- **Umfangreiche Wissensquellen**:
  Die beim NL–Parsing zugrundeliegenden Wissensquellen (Lexika, Grammatiken und Ontologie) sind sehr umfangreich und können auf relativ einfache Weise erweitert werden.

- **Erweiterbarkeit** der Wissensquellen durch deklarative Formalismen:
  Die Erweiterbarkeit ist eine wichtige Voraussetzung für die Anpassung des Systems an die jeweilige Umgebung. Dazu gehört auch die Adaption auf andere Sprachen.

- **Hohes Maß an Modularität**:
  Die Modularität ist eine wichtige Voraussetzung für die Erweiterung und für die Integration der Teilsysteme der Informationsextraktionskomponente.
- **Adaptierbarkeit auf andere Sprachen**:
  GETESS wird Deutsch und Englisch unterstützen.

Anpassung von SMES/Ontologie an die GETESS–Anforderungen Zunächst wurde die linguistische Komponente mit dem Wortschatz der im GETESS–Projekt zugrundeliegenden Domäne "Tourismus in Mecklenburg Vorpommern" trainiert. Zum **ersten Meilenstein** des Projektes (vgl. [BPW+98]) wurde eine lexikalische Abdeckung von 89% der gelesenen Worte erreicht.
Für die Ontologie–Komponente wurde ausgehend vom Ontobroker–Projekt [FDES97] in Karlsruhe ein Prototyp einer GETESS–konformen Ontologie erstellt. Dabei wurden die Schnittstellen zu den anderen GETESS–Komponenten beschrieben.

Zum **zweiten Meilenstein** (vgl. [MPP+99]) konnte die lexikalische Abdeckung bei SMES auf 100% gesteigert werden. Dabei wurden die Wissensquellen teilweise per Hand erweitert.
Die Ontologie wurde in Karlsruhe um weitere Konzepte erweitert und der Zugriff auf die Ontologie durch eine Inferenzmaschine ermöglicht.
Weiterhin wurde die Ontologie zur semantischen Analyse im natürlichsprachlichen Parsing–Prozeß integriert. Dabei wurden linguistische Lexika auf Ontologiekonzepte abgebildet und die Grammatiken um eine domänenspezifische Schnittstelle erweitert.

Der momentane Stand der Arbeiten an der NL-Komponente sieht das Einbinden spezieller Grammatiken (z.B. eine Adreßgrammatik) und das Spezifizieren natürlichsprachlicher Wissensquellen zur Behandlung englischsprachlicher Texte vor. Außerdem werden von den Arbeitsgruppen Karlsruhe und Saarbrücken gemeinsam Tools zur Aquisition und Pflege der domänenspezifischen Module entworfen.

3.2.2 Architektur des NL–Parsing

In Abbildung 3.2 ist die momentane Architektur des NL–Parsing dargestellt. Die Skizze zeigt den grundlegenden Aufbau. Die Architektur der Kernkomponente ist in [NBB+97] und [BDB+00] genau beschrieben.

Wie in Abbildung 3.2 zu sehen, beschreibt die Architektur ein verteiltes System. Es gibt einen oder mehrere Clients, die als Nutzer der NL–Analyse auftreten. Der Server umfaßt die Kernkomponente des NL–Parsings. Die Ontologie dient als semantisches Referenzmodell zur semantischen Analyse der zu parsenden Texte und ist standortunabhängig.
Nachfolgend werden die einzelnen Komponenten der Architektur vorgestellt:

- **SMES–Client**
  Der SMES–Client beinhaltet ausschließlich den Zugriff auf den Server. Er schickt ein Dokument an den Server und gibt das geparste abstract zurück.
- **SMES–Server**
  Bei Anfrage an den SMES–Server werden die folgenden Analyseprozesse durchlaufen.
  - **Text Tokenizer**
    In dieser Phase werden Textstrukturen separiert und als Token, auch teilweise normalisiert (z.B. Datumsangaben), ausgegeben.
  - **Morphologische/lexikalische Analyse**
    Die morphologische, lexikalische Analyse gibt für jedes erkannte Token ein Tripel der Form {Wortstamm, Flexion, Position im Text} aus. Weiterhin werden lexikalische Informationen gesammelt, um z.B. Mehrdeutigkeiten zu erkennen.
  - **Syntaktische Analyse**
    Für die syntaktische Analyse werden Chunk Parser eingesetzt. Ein Chunk Parser ist

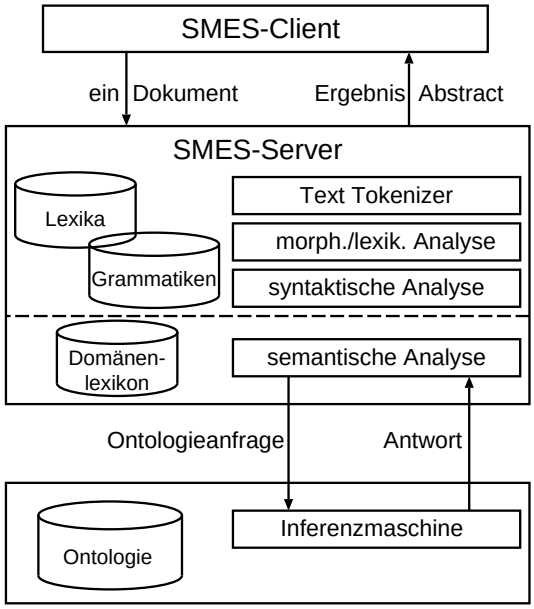

Abbildung 3.2: Architekturskizze des NL–Parsing–Vorgangs

nach [MPP+99] ein Parser mit mehreren flachen Grammatiken, die jeweils nur bestimmte Phrasen bearbeiten können. Es werden Satzfragmente bestimmt und ihre Beziehungen analysiert. Für jeden Satz wird ein sogenannter FST (finite state transducers [NBB+97]) generiert. Diese FSTs sind Repräsentationen für die jeweiligen Satzfragmente und dementsprechend abhängig von der zur Analyse verwendeten Spezialgrammatik.

– **Semantische Analyse**
Bei der semantischen Analyse werden bestimmte Phrasen mittels eines Domänenlexikons herausgefiltert. Das Domänenlexikon ist eine Abbildung der linguistischen Phrasen auf Ontologiekonzepte. Dabei werden Phrasen, die auf mehrere Konzepte abgebildet werden können, auch mehrfach im Domänenlexikon aufgenommen. Die betreffenden Phrasen werden dann mit dem Ontologiewissen des Domänenlexikons angereichert. Weiteres Ontologiewissen (und damit die semantische Information der Texte) wird durch eine Anfrage an die Ontologie bezogen.

- **Ontologie**
Bei Anfragen an die Ontologie wird durch eine Inferenzmaschine in der Ontologie nach der Antwort gesucht bzw. mittels der Ontologie berechnet. Eine typische Anfrage in diesem Zusammenhang ist die Frage nach der Beziehung zwischen zwei im NL–Parsing erkannten Konzepten.

Laut [BDB+00] hat das SMES–System einen deutschen Stammwortschatz von 120000 Einträgen. Die NL–Analyse eines Textes mit 400 Wörter benötigt nur ungefähr eine Sekunde.

Die abstract–Generierung Ausgangspunkt für die abstract–Generierung ist das Ergebnis der domänenspezifischen NL–Analyse der Texte. Zur Bildung eines abstract werden die durch das Domänenlexikon bestimmten Phrasen in Beziehung gesetzt. Dies geschieht durch eine Ontologieanfrage.
Technisch sind dazu die als endliche Automaten definierten Grammatiken in einen Erkennungsteil und einen Ausgabeteil unterteilt. Der Ausgabeteil wird durch einfache Erweiterungen um die Angabe domänenspezifischer Informationen ergänzt.

Entscheidend ist nun die Frage, welche der Phrasen bzw. Terme (das können innerhalb eines Dokumentes sehr viele sein) in Beziehung gesetzt werden. Dies wird mittels linguistischer Regeln und vor allem Heuristiken entschieden. Folgende positive und negative Heuristiken [BDB+00] werden derzeit angewendet:

- **Titel–Heuristik:**
  Kombination eines Terms aus dem Titel mit allen anderen im Text stehenden.
- **Satz–Heuristik:**
  Kombination aller Terme eines Satzes untereinander.
- **NP–PP–Heuristik:**
  Kombination von aufeinanderfolgenden NP– und PP–Phrasen miteinander.
- **Instanz–Heuristik:**
  Keine Kombination zwischen Termen mit dem gleichen Instanznamen.
- **Koordinations–Heuristik:**
  Keine Kombination zwischen Termen innerhalb gleichgestellter Nominalphrasen; z.B. beim Satz "Das Hotel hat einen Swimmingpool und eine Dachterrasse." wird "Swimmingpool" und "Dachterrasse" nicht in Beziehung gesetzt.

Beziehungen, die durch mehrere Heuristiken ausgedrückt werden können, werden zusammengefaßt und bekommen eine höhere Relevanz.

Als Ausgabeformat wurde eine XML–Darstellung gewählt. Dabei wurde die interne FST–Darstellung in das projektinterne Austauschformat XML überführt. Die Schnittstellendefinition wird über DTDs (Document Type Definition für XML) geregelt. Die XML–Ausgabe ist deklarativ und flexibel realisiert.

3.2.3 Funktionsweise der NL–Software

Wie in der Architektur beschrieben, ist die NL–Analyse als Client/Server–Prozeß realisiert. Der Server ist am DFKI Saarbrücken installiert und wird unter anderem im Rahmen des GETESS–Projektes weiterentwickelt. Der Server beinhaltet die gesamte natürlichsprachliche Textverarbeitung mit allen Lexika und Grammatiken. Die Ontologie und deren Inferenzmaschine gehören zum Arbeitspaket des AIFB Karlsruhe und sind dort installiert.

Der Server ist unter

```
let.dfki.uni-sb.de
```

auf einem bestimmten TCP–Port ansprechbar. Der SMES–Client greift über diesen Port auf den Server zu. Dazu wird die Software "parac", der SMES–Client, in der momentan vorliegenden Version 0.9 installiert. Man benötigt für die Software Perl (eine Skriptprogrammiersprache) ab der Version 5.004_04 und PerlTk ab der Version 8.00_012. Der Client besteht aus einem Perlskript und mehreren Perlmodulen. Es gibt neben dem Kommandozeilenmodus, auch die Möglichkeit per graphischer Oberfläche[1] mit dem Client zu arbeiten. Für die Einbindung des Client in den Gathering–Prozeß wird ausschließlich der Kommandozeilenmodus benötigt.
Nachdem man die Software installiert, konfiguriert und Umgebungsvariablen gesetzt hat, kann der Client benutzt werden.

Aufruf des Client Gestartet wird der Client mit folgendem Aufruf:

```
parac -m CALL [OPTIONS]
```

Dabei bedeutet "-m" der Kommandozeilenmodus, und "CALL" bezeichnet zum einen das Eingabeformat, und zum anderen können Ebenen des Textparsing ausgewählt werden (z.B. nur morphologische Analyse). Beispielsweise wird mit der folgenden "CALL"–Angabe die Verarbeitung von Strings mit allen Parsing–Ebenen ausgewählt:

[1] Aus diesem Grunde benötigt das Programm PerlTk.

```
'smes::fst-from-string ((:fst . :all-frags))'
```

Die Optionen ([OPTIONS]) sind folgendermaßen zu verstehen:

- -h <host> — Servername.
- -i <file> — Datei mit dem zu parsenden Text; wenn nicht angegeben, wird die Standardeingabe gelesen.
- -p <port> — TCP–Port des Servers.
- -t <type> — Auswahl des Ergebnistyps, z.B. 'text/plain' oder 'text/xml'.
- -v <level> — Angabe des debug level von '0' für keine Informationen bis '10' für maximale Ausgabe.

Parsing–Ergebnis Nachdem der Client gestartet und das zu parsende Dokument übergeben wurde, erhält man ein Ergebnis in Form einer Liste von jeweils zwei in Beziehung gesetzten Konzepten. Diese Liste wird in Form eines XML–Dokumentes generiert. Die Definition dieser Schnittstelle wird durch die folgende DTD beschrieben.

```
<!-- abstract-DTD -->

<!ELEMENT list       (tuple*)>
<!ELEMENT tuple      (fragment, fragment, constraint)>
<!ELEMENT fragment   (type, inst, name)>
<!ELEMENT constraint (#PCDATA)>
<!ELEMENT type       (#PCDATA)>
<!ELEMENT inst       (#PCDATA)>
<!ELEMENT name       (#PCDATA)>

<!ATTLIST tuple
          name       NMTOKEN     #REQUIRED
          rel        NMTOKEN     #IMPLIED>
<!ATTLIST constraint
          type       NMTOKEN     #REQUIRED>
```

Eine Erklärung dieser DTD wird das folgende Beispiel anhand eines Testdokuments geben.

3.3 Beispiel einer SMES–Anfrage

Im Anhang A ist ein Dokument der "www.all-in-all.de"–Domäne dargestellt. Dieses Dokument, als Original in HTML–Form vorliegend, wird für das SMES–Beispiel nun geparst. Dazu wird der HTML–Code dieser Seite mittels folgendem Aufruf an die NL–Analyse übergeben:

```
parac -m 'smes::pairs-from-html ((:fst . :all-frags))' -i 1127.htm >1127.xml
```

Es werden nun Ausschnitte der HTML–Seite und die zugehörigen Teilergebnisse des NL–Parsing betrachtet. Die Ergebnisse sind gekürzt und in einer Tabelle zusammengefaßt, da die Ausgabe der NL–Analyse für ein Dokument sehr umfangreich ist. Das vollständige abstract ist im Anhang B zu finden.

Die Tabelle 3.1 zeigt einige der gefundenen Ergebnisrelationen zwischen extrahierten Konzepten. Jedes Konzept wird als Tripel dargestellt. Dies ist der Typ, die Instanz und der Name des Konzeptes. Der Typ ist das zugehörige Konzept aus der Ontologie. Die Instanz wird durch Durchnumerierung der unterschiedlichen Instanzen eines Typs oder durch einen konkreten Instanznamen ausgedrückt. Der Name bezeichnet die Originalphrase aus dem Dokument. In der Tabelle sind Typ und Instanz zusammengefaßt. Die Relation bezeichnet den gefundenen Slot der Konzepte aus der Ontologie. In der letzten Spalte stehen die Erkennungsmerkmale, teilweise werden hier Heuristiken

Nr	Relation	Typ/Instanz 1	Name 1	Typ/Instanz 2	Name 2	Heu.
1	in_Stadt	Hotel (1)	hotel	Stadt	Rostock	th,sh,nph
2	-	Hotel (1)	hotel	Region	Mecklenburg	sh
3	-	Stadt	Rostock	Region	Mecklenburg	sh
4	in_Stadt	Hotel (2)	hotel	Stadt	Rostock	th
5	in_Stadt	Hotel (1)	hotel	Stadt	Schwerin	th
6	-	Hotel (1)	hotel	Tourist (1)	tourist	th
7	-	Hotel (1)	hotel	Radio (1)	radio	th
8	-	Region	Mecklenburg	Telefon(1)	telefon	th
9	-	Telefon (1)	telefon	Fernseher (1)	tv	sh
10	-	Telefon (1)	telefon	Radio (1)	radio	sh
11	-	Telefon (1)	telefon	Foen (1)	foen	sh
12	-	Fernseher (1)	tv	Radio (1)	radio	sh
13	-	Fernseher (1)	tv	Foen (1)	foen	sh
14	-	Radio (1)	radio	Foen (1)	foen	sh
15	-	Hotel (1)	hotel	Person (1)	person	th
16	-	Person (1)	person	Person (1)	40	card
17	-	Hotel (1)	hotel	Einbettzimmer (1)	einzelzimmer	th
18	-	Hotel (1)	hotel	Zweibettzimmer(1)	doppelzimmer	th
19	-	Hotel (1)	hotel	Mehrbettzimmer(1)	dreibettzimmer	th
20	-	Einbettzi. (1)	einzelzimmer	Zweibettzimmer(1)	doppelzimmer	sh

Tabelle 3.1: SMES–Anfrageergebnis der HTML–Seite aus Abbildung A.2 (Heu = Heuristiken, th = Title-Heuristic, sh = Sentence-Heuristic, nph = NP-PP-Heuristic, card = Kardinalbeziehung)

und teilweise sprachliche Beziehungen verwendet.

Anhand der Ergebnisse der Tabelle 3.1 werden nun die NL–Analyse von GETESS erklärt und die momentanen Probleme dieses Prozesses besprochen.
Es wurden gerade mal drei Relationen benannt, wobei immer dieselbe Relation gesetzt wurde. Die erste von den drei Relationen ist korrekt erkannt. Hier wurde das Hotel 1 richtig der Stadt Rostock zugeordnet. Bei der zweiten Relation (Nr. 4) wird ein zweites Hotel (Hotel 2) der Stadt Rostock zugewiesen. Wie im Anhang A zu sehen, beschreibt die HTML–Seite nur ein Hotel und damit ist diese Relation nicht korrekt. Die dritte Relation (Nr. 5) ordnet das Hotel 1 der Stadt Schwerin zu. Dieses nicht korrekte Ergebnis der Title–Heuristic entstand durch die Beziehung zwischen Hotel aus dem Titel und der Copyright–Zeile ganz unten (siehe Anhang A). Die erste Relation ist durch drei Heuristiken entstanden und somit besonders relevant.
Die Ergebnisse Nr. 2 und Nr. 3 sind nachvollziehbare Beziehungen, allerdings wurden hier keine Relationen erkannt. Bei Hotel erwartet man außerdem einen Hotelnamen. Die Namenserkennung ist noch unzureichend.
Direkte Beziehungen zwischen den Konzepten der Ergebnisse Nr. 6, Nr. 7 und Nr. 8 sind nicht erkennbar. Offenbar werden Beziehungen ausgewertet, die indirekt über andere Beziehungen einen Sinn ergeben. Beispielsweise könnte die Beziehung bei Nr. 7 von Hotel über Zimmer über Ausstattung zum Radio lauten.
Bei den Nummern 9 bis 14 sind die Konzepte Telefon, "Foen", Fernseher und Radio untereinander in Beziehung gesetzt. Die Beziehungen dieser innerhalb der Ausstattungsmerkmale eines Hotelzimmers gleichberechtigten Konzepte sind unnötig, da alle diese Merkmale an die Zimmer gebunden werden.
Nr. 15 ist ein Ergebnis ähnlich wie in den Nummern 6 bis 8. Eine direkte Beziehung zwischen Person und Hotel ist nicht erkennbar. Die nächste Beziehung (Nr. 16) ist eine Kardinal–Beziehung zwischen Person und der Zahl 40. Das Erkennen solcher Zahlen ist ein sehr interessanter Aspekt, allerdings existiert in der Ergebnisliste kein Bezug der 40 Personen zur Tagungs– bzw. Seminarkapazität.

Die drei Beziehungen Nr. 17 bis Nr. 19 der Ergebnisauswahl zeigen die korrekten Verbindungen zwischen dem Hotel und den Zimmern. Das Ergebnis Nr. 20 zeigt die von den Ausstattungsmerkmalen bekannten, unnötigen Verbindungen zwischen gleichberechtigten Konzepten, hier die einzelnen Zimmerarten.

Eine interessante Sicht auf die Ergebnisse ist die Verteilung auf die Heuristiken. Die NP–PP–Heuristic kommt nur einmal vor und kennzeichnet die einzig benannte, richtige Relation. Die Kardinalbeziehung kommt auch nur einmal vor und legt die Personenanzahl richtig fest. Die Sentence–Heuristic und die Title–Heuristic sind sehr oft vertreten, allerdings sind hier sehr viele falsche bzw. unsinnige Beziehungen gefunden worden. Bei der Sentence–Heuristic sind ungefähr 75% irrelevant oder falsch, und bei der Title–Heuristic sind es etwa 85% irrelevante bzw. falsche Beziehungen.

Die erkannten Konzepte sind mit Ausnahme von "Schwerin" korrekt, allerdings fehlen einige wichtige Konzepte, wie Zimmerpreise oder Anschrift.

3.4 Zusammenfassende Betrachtungen der NL–Analyse

Die NL–Analyse stellt die Kernkomponente der Informationsextraktion in GETESS dar. Die Kombination der linguistischen Analyse mit wissensbasierten Techniken ergibt ein Tool, welches in kurzer Zeit große Texte parsen und abstracts generieren kann.

Der Client parac des Analyse–Tools wurde in den Gathering–Prozeß integriert und liefert Ergebnisse in Form von XML–Files. Diese XML–Files werden pro Dokument angefertigt und sind damit exakt einem Dokument zuordenbar. Das abstract bildet eine Eigenschaft des Dokuments.
Aus dem vorherigen Beispiel resultiert, daß viele Beziehungen innerhalb der Dokumente nicht oder falsch erkannt werden. Geeignete Zusatzinformationen, wie sie im Kapitel 2 dargestellt wurden, können diese Fehlinterpretationen erkennen und verbessern helfen.
Als weiterer Punkt ergibt sich aus den Motivationen dieser Arbeit, daß verteilte Information zu mehreren Dokumenten pro abstract und Informationsballungen zu mehreren abstracts pro Dokument führen kann. Solche Problemstellungen sind mit dieser hier dargestellten NL–Analyse nicht oder nur unzureichend realisierbar.

Zur Modellierung dieser Daten ist zu sagen, daß die XML–Darstellung der abstract–Daten als Eigenschaft eines oder mehrerer Dokumente in die XML–Darstellung der Speicherkomponentenschnittstelle gut integrierbar ist. Einzelne Teilaspekte der NL–Analyse könnten für den Vorgang der Informationsextraktion innerhalb des Gathering von Bedeutung sein. Diese Möglichkeiten werden im nächsten Kapitel, bezüglich der Integration der Konzepte zur Informationsextraktion, betrachtet.

Kapitel 4

Konzeption der Informationsextraktion

Nachdem die verschiedenen, möglichen Datenkonzepte in Kapitel 2 und Kapitel 3 vorgestellt wurden, werden nun Integrationsmöglichkeiten zur Lösung der hier gestellten Problemstellungen der Informationsextraktion diskutiert.
Es werden zunächst einige bestehende konzeptionelle Ansätze und technische Umsetzungen untersucht. Daraufhin werden mögliche Konzeptionen vorgestellt und unter verschiedenen Aspekten genau betrachtet.

4.1 Relevante Forschungsansätze

In diesem Abschnitt werden Ansätze aus den Forschungsgebieten Information Retrieval, Data Mining bzw. Knowledge Discovery in Databases und Wrapperbau betrachtet. Aus den vielen verschiedenen existierenden Ansätzen wird hier eine kleine Auswahl, die für diese Arbeit besonders interessante Konzepte beinhaltet, vorgestellt. Die Ansätze werden kurz beschrieben und wichtige Aspekte herausgestellt.

4.1.1 Strudel

Das System Strudel [FFLS97] dient dem Webdaten–Management. Es nutzt ein semistrukturiertes Datenmodell, ähnlich dem OEM (siehe Kapitel 2.3). Es modelliert Webseiten aus einem Datenbank–Repository und stellt diese im Internet bereit.
Die für diese Arbeit interessante Informationsextraktion wird hier nicht vordergründig behandelt. Es können verschiedene Datenquellen in das System integriert werden. Strudel benutzt dafür Wrapper. Die Informationskombination wird mittels einer gemeinsamen Datenmodellierung innerhalb der Mediatorschicht erreicht. Die Datenmodellierung wird durch ein Graphenmodell realisiert.
Die Daten der Mediatorebene können durch eine Seiten–Definition (site graph) in einer HTML–Internetserver–Struktur präsentiert werden. Weiterhin wurde eine spezielle Anfragesprache StruQL (Strudel Query Language) für die Anfrage auf die Daten im site graph und der Mediatorschicht definiert.

4.1.2 Lore und TSIMMIS

Lore (Lightweight Object Repository) [MAG+97] ist zur Verwaltung semistrukturierter Daten an der Stanford University entwickelt worden. Es wurde vor allem für Daten mit sehr wenig Struktur konzipiert. Die Modellierung der Daten basiert auf dem OEM–Schema (siehe Kapitel 2.3). Das Modell läßt sich nach [AQM+96] auf das objektorientierte Datenmodell ODMG abbilden. Und die für Lore eingeführte Anfragesprache Lorel (Lore Language) ist nach [AQM+96] eine Erweiterung von OQL.
Lore vereint, ähnlich wie Strudel, verschiedene heterogene Datenquellen. Heterogene Daten werden,

wenn nötig, mittels Typumwandlungen in das interne Datenmodell eingebunden. Die Datenquellen werden durch Wrappertechniken angesprochen. Hierfür wurden Ansätze aus dem TSIMMIS–Projekt [CGMH+94] angewendet.
Das Ziel des TSIMMIS–Projekt (The Stanford IBM Manager of Multiple Information Sources) ist die konsistente Integration von heterogenen Datenquellen. TSIMMIS favorisiert einen möglichst automatischen bzw. halbautomatischen Zugang zu den Datenquellen. Die Wrapper–artige Architektur spricht von sogenannten Translatoren, die den Zugriff ermöglichen, wobei per Definition ein Generator solche Translatoren aufbaut. In der Mediatorebene, eine Mittelschicht, wird die Integration der Datenquellen durchgeführt. Dabei ist auch im geringen Maße Duplikaterkennung und -elimierung möglich. Diese Module können ebenfalls per Definitionen generiert werden.

Ein interessanter Aspekt bei Lore sind die DataGuides. Sie beschreiben und visualisieren die Daten und Informationen mittels Graphen. Durch Anfragen können diese Graphen iterativ aufgebaut und verfeinert werden.

4.1.3 NoDoSE

NoDoSE (Northwestern Document Structure Extractor) [Ade98] ist ein Tool zur halbautomatischen Extraktion von strukturierten und semistrukturierten Daten aus Textdokumenten. Das Paper zeigt die Funktionalität des Tools ausschließlich an HTML–Beispielen. Prinzipiell können beliebige strukturierte Daten analysiert werden.

Ausgangspunkt ist eine Menge ähnlicher Dokumente. Darauf wird eine Strukturextraktionskomponente durch Regeln definiert. Mittels einer graphischen Oberfläche und den Regeln erstellt der Nutzer ein Modell der Daten. Die Extraktionskomponente gibt eine Instanz des Modells für jedes Dokument aus. Aus den Modelldaten und den extrahierten Dokumentdaten können nun Grammatiken zur Wrappergenerierung, Reports oder Datenbankschemata und Datenbankdaten generiert werden.
Die Modellierung geschieht mit Hilfe des Extraktionstools und den Nutzer–definierten Daten. Das Modell für eine Dokumentgruppe wird durch einen Baum dargestellt. Jeder Knoten beinhaltet einen atomaren oder mengenwertigen Typ, einen relativen Offset, einen Label, eine AuthorID und einen confidenceValue zur Kennzeichnung korrekt geparster Knoten. Durch die mengenwertigen Knoten wird der Baum aufgespannt.

Die Begrenzung dieses Ansatzes spiegelt sich in der Annahme der gleichen Struktur der zu verarbeitenden Dateien wider.

4.1.4 ARANEUS

Im ARANEUS–Projekt [MAM+98] wurden Tools zum Datenmanagement im Internet entworfen. Dabei wurden auch Fragestellungen der Datenmodellierung und der Informationsextraktion von HTML–Seiten betrachtet.
Bei den Betrachtungen zum Internet wurde in ARANEUS herausgestellt, daß, obwohl das Web im allgemeinen sehr unstrukturiert ist, besonders die großen Webserver gut strukturiert und mittels Datenbanktechniken gut zu beschreiben sind. Trotzdem stellt man in ARANEUS die Informationsextraktion als ein schwieriges Problem dar.
Zur Modellierung wird das ADM–Datenmodell, eine erweiterte Untermenge vom ODMG, verwendet. Mittels diesem Modell werden hauptsächlich per Hand die Web–Seiten beschrieben. Logisch homogene HTML–Seiten werden durch ein Seiten–Schema zusammengefaßt.
Um die ADM–Strukturen mit Daten zu füllen, werden Wrapper eingesetzt, die durch eine prozedurale Sprache (EDITOR) beschrieben werden. Trotz der Probleme einer prozeduralen Programmierung, wie viele Programmierdetails und mangelnde Flexibilität, wird dieser Ansatz einem herkömmlichen Parser mit kontextfreier Grammatik vorgezogen. Eine Verbesserung der Probleme wurde mit der MINERVA–Sprache erreicht. MINERVA bietet einen Kompromiß zwischen dem prozeduralen Ansatz und dem Parseransatz.

4.1.5 Zusammenfassung und Bewertung

Die hier vorgestellten Ansätze gehen vor allem in der Modellierung ähnliche Wege. Viele Ansätze benutzen OEM oder OEM–konforme Modelle, um Web–Daten darzustellen. Der wesentliche Vorteil besteht darin, daß die Struktur und die Inhalte mit ihren Beziehungen in einem Modell beschrieben und angefragt werden können.

Das Einsammeln und Analysieren der Daten geschieht meist mit einem vordefinierten Schema. Dazu werden häufig Wrappertechniken verwendet. Dabei kommen Werkzeuge, wie z.B. im TSIMMIS–Projekt [CGMH+94] oder im W4F [SA99] (Sammlung von Tools zur Wrapper–Generierung) entwickelt, zum Einsatz. Ansätze zur Informationsextraktion mittels linguistischer oder domänenspezifischer Analyse sind nur in vereinzelten, speziellen Umgebungen zu finden. Eine Kombination derartiger Techniken mit einer umfangreichen Strukturanalyse, wie sie in GETESS angestrebt wird, ist momentan nicht zu finden.
Zusammenfassend ist zu sagen, daß die meisten Ansätze mittels der Wrappertechnik direkt von der Struktur auf die Inhalte schließen. Die momentan vorhandenen Internet–Informationen sind allerdings nicht immer an eine erkennbare Struktur gebunden, so daß diese Technik nur eingeschränkt eingesetzt werden kann.

Viele Ansätze benutzen eine sogenannte Mediatorschicht als Grundlage für die jeweilige, konkrete Anfragesprache. Diese Schicht beschreibt die eventuell heterogenen Daten gemäß der Modellierung in einer einheitlichen Sicht. Eine solche einheitliche Sicht wird in GETESS durch den Einsatz der Anfragesprache IRQL ([HP99] und [HP00]), die auf SQL–basierte und Information Retrieval–Systeme zugreifen kann, erreicht.

4.2 Konzeptionelle Architektur

Für die Informationsextraktion im GETESS–System wird nun ein Architekturvorschlag diskutiert. Dabei wird ein allgemeiner Überblick über die benötigten Komponenten und deren Funktionalität gegeben.

4.2.1 Architektur

Ausgangspunkt für die folgende Architekturbeschreibung ist die vorgestellte Architektur des Gathering–Prozesses in Kapitel 1.3. Diese Architektur wird mit Komponenten erweitert, um die in Kapitel 2 und 3 vorgestellten Daten zu integrieren und zu verarbeiten.
Abbildung 4.1 zeigt eine allgemeine Architektur zur Informationsextraktion in GETESS.

In der Abbildung sind zwei Hauptkomponenten dargestellt. Der Gatherer beschreibt die Agenten–gesteuerte Sammelkomponente. Dazu gehören vor allem Angaben zu den zu durchsuchenden Seiten (Sammelvorbereitung), das Einsammeln der Dokumente und das Summarizing zur Auswertung von Gathering–relevanten Daten. Vom Summarizing aus wird die Datenaufbereitung gestartet und die gesammelten Dokumente und die bis dahin extrahierten Metadaten übergeben.

Die zweite große Komponente ist die Datenaufbereitung und Datenanalyse. Hier werden die gesammelten Daten und Dokumente analysiert, verarbeitet und anschließend für die Speicherung aufbereitet. Ein Controller–Baustein steuert die Informationsextraktion. Für die Bearbeitung der Daten sind der Controller–Komponente einige Analysewerkzeuge und Datenaufbereitungs–Tools angegliedert. Die abschließende Datenaufbereitung erzeugt Datensätze pro Informationspaket für die Speicherung in einer Speicherkomponente.

4.2.2 Komponentenüberblick

Ausgehend von der Architektur werden nun Teilkomponenten der Datenaufbereitung und -analyse im Überblick vorgestellt. Die Tabelle 4.1 zeigt die einzelnen Komponenten, die notwendigen Daten, die resultierenden Daten und eine kurze Beschreibung des Prozesses.

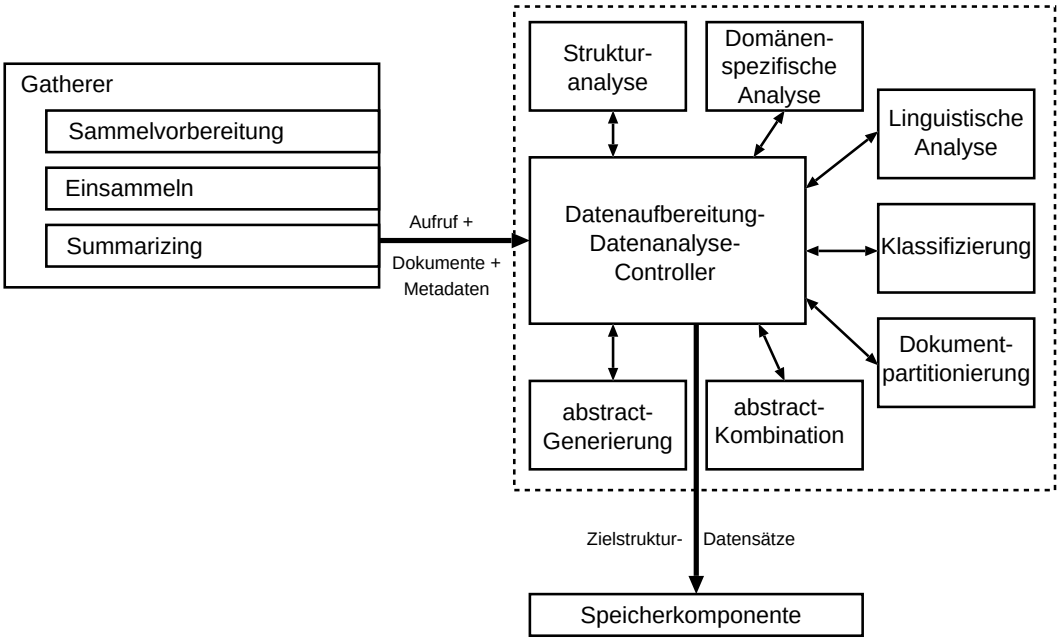

Abbildung 4.1: Allgemeine, konzeptionelle Architekturskizze zur Informationsextraktion

Die ersten drei Komponenten in der Tabelle 4.1 sind für die strukturelle, linguistische und domänenspezifische Analyse verantwortlich. Es werden hier grundlegende Daten extrahiert, die für die weitere Informationsextraktion wesentlich sind.
Durch geschicktes Partitionieren der Dokumente können nicht zusammengehörige Informationen separiert werden. Dabei werden mögliche irrelevante Daten und besonders relevante Daten strukturell getrennt. Die Klassifizierung ordnet die Seiten bzw. Seitenteile in eine bestimmte funktionelle und eine domänenspezifische Hierarchie ein. Zu den Dokumentbestandteilen der Teilbäume werden mit Hilfe der vorangegangenen Daten abstracts gebildet, die dann zu sinnvollen Komplett–abstracts kombiniert werden.

Die hier genannten Prozesse werden im nächsten Abschnitt 4.3 genau betrachtet, wobei die Prozesse zur Informationsextraktion hier im Vordergrund stehen.

4.2.3 Weitere Aspekte

An dieser Stelle nicht betrachtet sind weitere, für das GETESS–Suchsystem wichtige Konzepte, wie Verteilungsaspekte, Ranking oder Relevance Feedback, Einbindung externer Datenbanken und die Unterstützung mehrerer Sprachen. Diese Aspekte haben zwar einen Einfluß auf die Realisierung des Gatherings, werden die Qualität der Informationsextraktion allerdings nicht grundlegend verbessern, da die zugrundeliegenden Konzepte keine Inhaltsbeschreibungen für eine Informationsextraktion vorsehen.

Die Unterstützung mehrerer Sprachen spiegelt sich in unterschiedlich sprachlicher Repräsentation der Inhalte wider. Einen Ansatz zur Erkennung der Sprache eines Dokumentes ist in [DG00] für GETESS konzipiert und umgesetzt worden. Nach dem Sammelprozeß würde eine Einordnung in die Sprache sinnvoll sein. Die folgenden inhaltserschließenden Prozesse würden dann sprachabhängig die jeweiligen Analysen durchführen.

Der Verteilungsaspekt ist gerade für die Parallelisierung des Sammel- und Analyseprozesses interessant. Die Parallelisierung des Sammelprozesses ist dabei relativ unproblematisch. Die in dieser

Komponente	*benötigte Daten*	*resultierende Daten*	*Prozeßbeschreibungen*
Strukturanalyse	Dokument, Dokumenttyp–spezifische Daten	Struktur–Graph	Analyse der Dokumentstruktur und Speichern des resultierenden Strukturgraphen
Linguistische Analyse	Dokument, linguistische Lexika und Grammatiken	linguistisch aufbereitete Dokumentdaten	Extraktion und Speichern linguistischer Informationen
Domänenspezifische Analyse	Dokumentdaten, Ontologie–Daten	domänenspezifische Konzeptdaten	Speichern der extrahierten domänenspezifischen Konzepte und Zuordnung zur Struktureinheit des Dokumentes
Dokumentpartitionierung	Dokumentdaten, bisherige Analyseergebnisse	separate Dokumentteile	Dokument kleiner strukturieren
Einteilung, Klassifizierung von Dokumenten und Dokumentteilen	Strukturdaten, domänenspezifische Daten, Ontologie	aufgeschlüsselter, abstrahierter Struktur–Graph, Klassen	funktionelle und domänenspezifische Klassen für Dokumente bzw. Dokumentteile ermittteln
Teil–abstract–Erzeugung	Dokumentdaten, Strukturdaten, domänenspezifische Daten, Ontologie	abstract–Daten	Teil–abstracts als Zwischenergebnis
Kombination der abstract–Daten	abstract–Daten, Linkstrukturen, Ontologie	kombinierte abstracts	Aufbau der Ergebnis–abstracts als Hauptanteil der Zielstruktur

Tabelle 4.1: Überblick über die Analysewerkzeuge mit ihren Input– und Output–Daten.

Arbeit diskutierte Informationsextraktion geht von teilweise zusammenhängenden und zusammengehörigen Daten aus. Es ist zu klären, welche zu sammelnden und zu analysierenden Daten verteilt betrachtet werden können und wie die verteilten Analyseergebnisse wieder zusammengefaßt werden.

Zur Einbindung externer Datenbanken sind Betrachtungen im Projekt SWING [LDHM97] und [HMW99] angestellt worden. SWING erweitert die Indexerstellung einer herkömmlichen Suchmaschine mit keywords der einzubindenden Datenbank und stellt einen Suchmaschinen–konformen Zugriff auf die Datenbank bereit.

Ranking und Relevance Feedback sind Konzepte, die über die Informationsextraktion hinaus konzipiert und realisiert werden und deshalb in einer globaleren Betrachtung zu bearbeiten sind.

4.3 Konzeptionelle Beschreibung der Prozesse

Die nachfolgend diskutierten Komponenten aus der Architektur werden teilweise zu einem Prozeß zusammengefaßt, wenn sie inhaltlich zusammengehörig und aufeinanderfolgend sind. Das Hauptaugenmerk liegt auf der Konzeption der Extraktionsprozesse und Kombinationsprozesse.

4.3.1 Der Sammelprozeß

Der Sammelprozeß umfaßt die Sammelvorbereitung, das eigentliche Einsammeln und das Summarizing.

Vorbereiten des Sammelprozesses Zur Vorbereitung des Gathering gehört mindestens einmal die Angabe der Dokumente oder der Dokument–Server, die durchsucht werden sollen. Da in diesem Ansatz von einer Einschränkung der Suchmaschine auf ein oder mehrere spezielle Wissensgebiete ausgegangen wird, sind die zu durchsuchenden Seiten explizit auszuwählen. Bei der Auswahl werden die Seiten zumindest oberflächlich durch den Gathering–Administrator begutachtet. Informationen, die bei dieser Begutachtung gewonnen werden, können wichtige Aspekte beinhalten, die eine maschinelle Informationsextraktion nicht bzw. kaum erkennen kann. Zu diesen Informationen gehören eine gewisse Einordnung in die spezielle Thematik (z.B. Hotelangebote, Freizeitangebote oder eine allgemeine Touristikseite), Angaben über Größe, Güte und Tiefe bis hin zu erkannten Strukturen.

Als Ziel dieses ersten Prozesses wird eine Grundstruktur für jeden eingetragenen Dokument–Server oder jedes eingetragene Dokument angelegt. Die ersten Daten werden als Metadaten in die jeweilige Struktur eingeordnet. Die Modellierung der Metadaten wird dem Administrator vorgegeben. Sie sollte aber flexibel sein.

Einsammeln der Seiten Das Einsammeln der Seiten übernehmen Agenten, die die Dokument–URL entgegennehmen und nach dem Einsammeln die gesammelten Daten als File ablegen. Um alle Dokumente eines Dokument–Servers zu durchsuchen, werden die Linkstrukturen innerhalb dieses Servers verfolgt.

Hauptaufgabe dieses Prozesses ist es, die gesammelten Dokumente in die Dokument–Server–Struktur einzuordnen und geeignet temporär abzulegen.

Summarizing Dieser erste Analyse–Prozeß extrahiert Metadaten, die vor allem für das Gathering benötigt werden. Dazu gehört in erster Linie das Extrahieren der Links als Input für die Sammelagenten. Das Summarizing ist dokumenttypabhängig. Der Dokumenttyp wird mittels eines Dokumenttyp–Analysetools bestimmt.
Linkstrukturen werden nicht von allen Dokumenttypen unterstützt. Sämtliche Dokumenttypen können aber als Link in Link–fähige Dokumente eingebunden werden und stellen somit zumindest Blätter im Link–Graph dar.

Für die Informationsextraktion ist die Linkstruktur ebenfalls wichtig. Die extrahierten Linkstrukturen werden als Metadaten der Dokument–Server abgelegt. Diese Linkstrukturen werden als OEM–Daten gespeichert (siehe Kapitel 2.3).
Weitere Metadaten stellen der Zeitpunkt des Einsammelns, die Refresh– und Delete–Zeit und die Dateigröße dar. Mit diesen ersten Daten zu den Dokumenten wird für jedes Dokument eine Struktur angelegt, in der diese Dokumenteigenschaften und die weiteren Extraktionsergebnisse eingetragen werden.

4.3.2 Strukturelle Analyse

Die strukturelle Analyse ist dokumenttypabhängig. Der Dokumenttyp ist seit dem Summarizing bekannt. Die strukturelle Analyse beinhaltet bei textuellen Dokumenttypen eine Strukturanalyse bzw. syntaktische Analyse, sowie die Extraktion wichtiger Metadaten. Bei nicht–textuellen Daten, wie Bilder oder Applikations–Code, sind spezielle Analysetools anzuwenden. Im allgemeinen ist eine derartige Analyse nicht–textueller Daten sehr aufwendig und das Analyseergebnis ist in vielen Fällen für die inhaltliche Informationsextraktion nicht relevant.

HTML–Strukturanalyse Als Ergebnis der HTML–Strukturanalyse wird eine Baumstruktur angestrebt, die als Repräsentant der HTML–Seite fungiert und die wesentlichen Strukturinformationen vorhält.
Wie in Kapitel 2.3, der Modellierung der Konzepte, besprochen, wird eine OEM–Darstellung der HTML–Seite angestrebt.

Die HTML–Darstellung nach der HTML4–Norm [Tol97] besteht aus Tags, die Attribute enthalten und Text umschliessen können. Es gibt öffnende und schließende Tags. Einige dieser Tags können auch alleine, ohne schließende Tags, vorkommen. Die Tags bestimmen das Aussehen und die Struktur der darzustellenden Informationen und sind somit für die Informationsextraktion besonders interessant.
Die Abbildung der HTML–Seite auf die Baum–Darstellung wird nun vorgestellt.
In den Strukturbaum aufzunehmende Elemente sind:

- **Strukturelemente:**
  `<HTML>`, `<HEAD>`, `<BODY>`, `<FRAME>`, `<BR>`, `<HR>`, `<P>`, `<H1..7>`
- **Head–Elemente:**
  `<BASE>`, `<META>`, `<TITLE>`, `<ADDRESS>`, `<LINK>`, `<ABBR>`
- **Tabellen:**
  `<TABLE>`, `<CAPTION>`, `<TD>`, `<TR>`, `<COL>`, `<COLGROUP>`, `<TH>`, `<TBODY>`, `<THEAD>`, `<TFOOD>`
- **Textformatierungen**
  `<BIG>`, `<SMALL>`, `<B>`, `<STRONG>`, `<I>`, `<EM>`, `<U>`, `<S>`, `<TT>`, `<PRE>`, `<CITE>`, `<Q>`, `<SAMP>`, `<KBD>`, `<VAR>`, `<DFN>`, `<SUB>`, `<SUP>`, `<CENTER>`, `<DIV>`, `<SPAN>`, `<STYLE>`, `<BLOCKQUOTE>`, `<BDO>`, `<DEL>`
- **Links, Bilder:**
  `<A>`, `<IMG>`, `<AREA>`, `<MAP>`
- **Frame–Elemente:**
  `<FRAMESET>`, `<IFRAME>`, `<NOFRAMES>`
- **Liste:**
  `<OL>`, `<UL>`, `<LI>`, `<DIR>`, `<MENU>`, `<DL>`, `<DD>`, `<DT>`

Die Strukturelemente geben Aufschluß über den grundsätzlichen Aufbau und die wesentlichen Hervorhebungen der Seite. In den Head–Elementen sind beschreibende Eigenschaften der Seite zu finden. Tabellenstrukturen werden neben den eigentlichen Tabellenformatierungen oft für die grobe Seitenstrukturierung verwendet und sind demnach ähnlich interessant wie die Strukturelemente. Die Texthervorhebungen können einzelne Worte oder Wortgruppen, ganze Absätze bis hin zum Dokument näher beschreiben. Die Links und Bilder sind mit ihrer Lokalität interessant, an welchen Textblöcken ausgelagerte Informationen liegen könnten. Bilder bestimmen auch wesentlich das Erscheinungsbild und sind ein wichtiges Strukturierungsmittel. Frames bieten eine Möglichkeit Informationen auf mehrere Dateien zu verteilen und Dokument–übergreifende Navigation einer Internet–Präsentation zu unterstützen. Die Listen sind wie die Tabellen nicht nur auf kleine Informationseinheiten beschränkt, sondern können auch wesentliche Dokumentteile strukturieren.
In die Baum–Darstellung nicht aufgenommen werden Formular– und Programmier–unterstützende Tags, da sie keinen oder nur geringen Nutzen für die Informationsextraktion haben.

Je nach Anwendung kann die Auswahl der relevanten Tags konkretisiert werden.

XML–Analyse XML und die dazugehörigen Struktur– und Style–Sprachen sind verhältnismäßig junge Konzepte, wobei einige Aspekte zur Zeit nur als Vorschlag dem XML–Gremium des W3C (World Wide Web Consortium) vorliegen.
Die Strukturdefinitionen werden in der DTD (Data Type Definition) abgelegt. Diese Definition muß nicht zwangsläufig vorliegen. Wenn eine DTD vorliegt, vereinfacht dies die Strukturanalyse, da man einen Strukturgraph schon vorgegeben hat. Die DTD entbindet allerdings nicht von der linguistischen Analyse und von der domänenspezifischen Analyse.
Wenn keine DTD vorliegt, ist die Strukturanalyse ähnlich der HTML–Analyse mit dem Unterschied, daß man es nur mit strukturellen Tags unbekannter Semantik zu tun hat.

Die Möglichkeiten die Semantik zwischen Strukturbezeichnern und Layoutbezeichnern zu vermischen, wie es in HTML häufig der Fall ist, kann mit XML wahrscheinlich ebenso erfolgen. Dies wäre auf der Ebene der DTDs und Stylesheets durchaus möglich.

4.3.3 Linguistische Analyse

Die linguistische Analyse normiert mit Hilfe von Lexika und Grammatiken die einzelnen Wortphrasen und stellt linguistische Beziehungen heraus (siehe Kapitel 3). Die normierten Wortphrasen bilden die Grundmenge der möglichen, relevanten, domänenspezifischen keywords.
Das Ergebnis dieser Analyse umfaßt neben den normierten Wortphrasen auch linguistische Beziehungen. Weiterhin werden durch Spezialgrammatiken für beispielsweise Eigennamen, Zeitangaben und Adressen normierte, extrahierte Daten geliefert.

4.3.4 Domänenspezifische Analyse

Bei dieser Analyse geschieht die eigentliche keyword–Extraktion und damit die inhaltliche Analyse. Die keywords werden domänenspezifisch extrahiert, wobei bestimmte Worte oder Phrasen auf Konzepte der Domäne abgebildet werden. Das bedeutet, daß bestimmte Worte oder Phrasen, die eine Beziehung zum vorgegebenen Wissensgebiet haben, erkannt und auf die Konzepte der Wissensrepräsentation, z.B. einer Ontologie, abgebildet werden. Die Worte und Phrasen können durch die linguistische Analyse gewonnen werden. Die linguistische Analyse hat den Vorteil, daß jegliche Wortformen eines keywords erkannt und abgebildet werden können.
Die domänenspezifischen keywords werden zusammen mit ihrem Auftreten und dem zugehörigen Konzept der Domäne gespeichert. Die Position der keywords ermöglicht eine Einordnung in den jeweiligen Textteil.

Die keywords aus den teilweise vorhandenen META–Tags (beispielsweise die Anwendung in [LDHM97]) sind für eine domänenspezifische Information unbrauchbar. In diesen META–Tags werden meist nicht nur spezifische Charakteristiken der jeweiligen Seite abgelegt, sondern auch allgemeine Daten und für die Anwendung irrelevante Daten. Beispielsweise sind in der Hotel–Beispielseite von Anhang A keywords wie Reservierung, Küste, Tagung und Tourismus enthalten. Für eine Information Retrieval–Anfrage sind diese Begriffe gut geeignet, für die Informationsextraktion können diese Begriffe eher verwirrend wirken. Solche allgemeinen Beziehungen, wie zwischen Hotel und Tourismus, sind ohnehin in der Ontologie schon verankert. Ähnlich sieht es mit den anderen META–Tags aus.
Diese META–Tags werden an dieser Stelle nicht als keywords in die Informationsextraktion aufgenommen.

4.3.5 Dokumentpartitionierung

Die Dokumentpartitionierung versucht das Dokument soweit aufzuteilen, daß möglichst keine unabhängigen Informationspakete in einem Dokumentteil mehr auftreten können. In diesem Zusammenhang ist es sinnvoll, irrelevante Bestandteile, die die Informationsextraktion verfälschen könnten, herauszunehmen.

Separieren der Struktur Das Einteilen der Struktur in kleine Unterstrukturen geschieht hauptsächlich auf Basis der gefundenen keywords bzw. Konzepte. Zu viele Konzepte innerhalb einer Struktureinheit erfordern eine Aufspaltung in weitere Teilstrukturen. Dieses Vorgehen ist sinnvoll, da bei möglichst kleinen Strukturen mit wenigen Konzepten eine Vermischung von unabhängigen Informationspaketen nahezu ausgeschlossen werden kann.
Inwieweit nun diese Aufspaltung in Unterstrukturen fortzuführen ist, kann mittels folgender zwei Heuristiken festgelegt werden. Weitere Heuristiken können vom jeweiligen Anwendungsszenario aus bestimmt werden.

1. Wenn mehr als n Konzepte in einer Teilstruktur, dann Aufspalten der Teilstruktur in zwei oder mehr Teile der darunterliegenden Strukturhierarchieebene

2. Aufspaltung beenden bei Aufspalten von Satzstrukturen oder wenn keine weitere Hierarchieebene

Welche Strukturen aufgespalten werden, ist Dokumenttyp–abhängig. In HTML–Dokumenten sind besonders die Strukturelemente, die Head–Elemente, Tabellen– und Listelemente sowie einige Textformatierungselemente (siehe oben) für das Aufspalten interessant.

Entfernen irrelevanter Daten Das Entfernen der irrelevanten Daten (bzw. Teile davon) kann die eigentliche Informationsextraktion wesentlich verbessern. Die Menge der irrelevanten Daten ist allgemein die komplementäre Datenmenge der relevanten Daten. Da diese Unterscheidung mittels automatischer Mechanismen nicht entscheidbar ist (aufgrund der Komplexität der natürlichen Sprache), wird hier ein semiautomatischer Ansatz favorisiert.
Dokumentteile, die keine domänenspezifische Konzepte beschreiben, werden sofort aus der weiteren Bearbeitung entfernt. Weitere Dokumentteile, in denen zwar Konzepte und damit keywords gefunden wurden, können trotzdem für die Informationsextraktion verfälschende Daten beinhalten. Diese Dokumentteile können mittels einer syntaxabhängigen Irrelevanzbeschreibung pro Dokumenttyp und Herkunfts–Server bzw. auch pro konkrete Seite entfernt werden.

4.3.6 Klassifikation

Die Klassifizierung wird für jedes Dokument vorgenommen. Für jedes Dokument wird zum einen das in der Domänenhierarchie der Ontologie am höchsten eingestufte Konzept, das per extrahiertem keyword gefunden wurde, als Klasse bestimmt.
Zum anderen wird eine Klasse aufgrund der Funktion innerhalb der Serverstruktur des Informationsanbieters bestimmt. Hier wird zwischen Übersichtsseiten, Hauptkonzeptseiten und Unterkonzeptseiten unterschieden.

Klassifizierung nach Ontologiekonzepten Aus allen keyword–erzeugten Konzepten werden die in der Baumhierarchie der Wurzel am nächsten liegenden Konzepte herausgesucht, die diese Klasseneinteilung bestimmen. Hierbei können möglicherweise mehrere Klassen zutreffen.
Der Algorithmus zur Erkennung der übergeordneten Konzepte kann folgendermaßen aufgebaut werden:

- erstes Konzept von n in Zielmenge
- k–tes Konzept mit jedem Konzept der Zielmenge Beziehung prüfen:
  wenn Konzept k unterhalb des Konzeptes der Zielmenge (bzw. eine hierarchische Unterbeziehung hat), k nicht in Zielmenge, weiter mit $k+1$
  wenn Konzept k oberhalb des Zielkonzeptes (bzw. eine hierarchische Oberbeziehung hat), Zielkonzept aus Zielmenge löschen
  wenn Zielmenge bis Ende durchlaufen, k in Zielmenge
- Wiederholen des 2. Schritts bis alle n Konzepte durchlaufen.

Das Erkennen der möglichen hierarchischen Beziehung zweier Konzepte, kann mittels Analyse der Domänenhierarchie und der gerichteten Pfade in der Ontologie erreicht werden, wobei "is a"– und "part of"–Beziehungen interessant sind.
Die Konzepte der Ontologie stellen die potenziellen Klassen dar. Aufbau und Funktionalität sind unter anderem in [MPP+99] beschrieben.
Bei einigen Dokumenttypen können auch bestimmte strukturell gekennzeichnete Textteile zur Gewinnung der Klasse genutzt werden.
Die Anzahl der gefundenen Klassen ist unter anderem vom Aufbau der Ontologie abhängig.

Klassifizierung nach Funktionalität Die Erkennung der Funktionalität geschieht durch Anwenden einiger Heuristiken, die aus Strukturinformationen und Ontologie–Konzepten bestehen.

Die hier favorisierte Hierarchie sieht folgendermaßen aus:

1. **Übersichtsseite:**
   Eine Übersichtsseite besteht aus einer Menge von Konzepten, wobei einzelne Instanzen der Konzepte unmittelbar mit einem Link in Verbindung gebracht werden können. Den einzelnen Instanzen können keine bzw. nur sehr wenige Unterkonzepte zugeordnet werden. Beispielsweise hat ein Dokument mit Hotels aus Rostock eine Liste der Hotelnamen plus je eine Adresse und je einen Link zu weiteren Informationen.
   Als weitere Unterteilung stellt sich die Anzahl der unterschiedlichen, aufgezählten Konzepte dar.
   (a) ein Konzept
   (b) verschiedene Konzepte
2. **Hauptkonzeptseite:**
   Diese Seitenart besteht aus genau einem übergeordneten Konzept (eventuell zwei oder maximal drei) mit weiteren Unterkonzepten, wie z.B. eine Hotelseite, bei der das Hotel vorgestellt wird und je ein Link zu weiteren Seiten mit verschiedenen Zimmern und den Preisen existiert. Die Unterkonzepte können teilweise mit Links verbunden sein. Daraus ergibt sich die weitere Unterteilung.
   (a) Unterkonzepte durch Verlinkung
   (b) Unterkonzepte ohne Verlinkung
3. **Unterkonzeptseite:**
   Die Unterkonzeptseite besitzt ein oder mehrere eingehende Links von Seiten mit höherem Konzept. Auf einer solchen Seite können mehrere Konzepte vorhanden sein. Eine solche Seite stellt zum Beispiel eine Zimmerbeschreibung dar, die von der dazugehörigen Hotelseite gelinkt wurde.

4.3.7 abstract–Generierung

Die abstract–Generierung geschieht nun mit Hilfe der in den vorhergehenden Analyseschritten gewonnenen Daten. Mit Hilfe der Ontologie werden die vielen einzelnen Strukturen zu wenigen großen abstracts kombiniert.

Aufbau der Teil–abstracts Im Vordergrund stehen die gesammelten Konzepte pro kleinster separierter Struktureinheit. Diese Konzepte innerhalb einer Struktureinheit werden nun mittels Ontologieanfragen in Beziehung gesetzt. Die Beziehung wird durch den kürzesten Pfad des Ontologiegraphen ausgedrückt. Ideal sind direkte Pfade, da diese Beziehungen konkret beschreibbar sind.
Die Kombination der Konzepte kann durch linguistische Heuristiken auf den ursprünglichen, natürlichsprachlichen Phrasen unterstützt werden.

Kombination der abstracts Bei diesem Schritt wird versucht, die zusammengestellten Konzepte der einzelnen Struktureinheiten mit den Konzepten der umliegenden Struktureinheiten in Beziehung zu setzen. Die Verknüpfung der Konzeptstrukturen kann durch Anhängen oder durch Verschmelzen entstehen.

Für die Kombination benötigt man Ontologiewissen und strukturspezifische Heuristiken.
Mittels strukturspezifischer Heuristiken werden die möglichen, zu kombinierenden Teil–abstracts bestimmt. Neben anwendungsspezifischen Heuristiken sind folgende, allgemeine Heuristiken anwendbar:

- Kombiniere alle Teil–abstracts der direkten, unmittelbaren Umgebung miteinander.

- Kombiniere durch Link verbundene Teil–abstracts miteinander.

Das Gerüst für das abstract liefert die Ontologie. Je nachdem, wie umfangreich die Ontologie und damit das Anwendungsszenario ist, desto schwieriger ist das Verknüpfen der Konzepte. Hilfreich sind dabei ein oder mehrere Einstiegspunkte vom Dokument in die Ontologie. Die generierte Ontologieklasse wäre dazu beispielsweise geeignet. Von einem solchen Einstiegspunkt aus kann dann versucht werden, alle weiteren Dokumentkonzepte anzuknüpfen.
Bei der Kombination werden bei gleichen Konzepten zunächst die Instanzen verglichen. Sind gleiche Instanzen gefunden worden, werden die Teil–abstracts an genau diesen Stellen verknüpft. Wurden unterschiedliche Instanzen bei gleichen Unterkonzepten (das heißt nicht bei den Toplevel–Konzepten der jeweiligen Teil–abstracts) gefunden, werden Listen von Instanzen zum jeweiligen Konzept gebildet. Danach wird noch versucht neue Ontologie–Relationen zwischen den Konzepten der beiden Teil–abstracts zu finden.
Kardinalitäten der Ontologierelationen und eine Numerus–Bestimmung (Plural/Singular) der domänenspezifischen keywords unterstützen die Kombination. Diese Angaben bestimmen die mögliche Anzahl von Instanzen der Relation.

4.4 Beschreibung der Zielstruktur

Abschließend zu den konzeptionellen Betrachtungen zur Lösung der Informationsextraktion im GETESS–Projekt wird die Zielstruktur betrachtet. Die Zielstruktur ist das Ergebnis des Sammelprozesses und der Informationsextraktion und stellt die Übergabestruktur zur Speicherkomponente dar.
Die exakte Strukturbeschreibung ist von der konkreten Speichersituation abhängig und sollte daher flexibel sein. Es wird eine XML–Darstellung favorisiert, da damit eine flexible Umwandlung in verschiedene, konkrete Zielstrukturdarstellungen gewährleistet werden kann.

Dokumente und abstracts Bisher wurde für jedes Dokument genau ein abstract gebildet. Durch die hier favorisierte Informationsextraktion gibt es weitere mögliche Abbildungen von den Dokumenten auf abstracts.

1:1 ein Dokument → ein abstract
Bsp.: Hotelseite wie in Anhang A

1:n ein Dokument → mehrere abstracts
Bsp.: Übersichtsseite mit Beschreibung mehrerer Hotels

n:1 mehrere Dokumente → ein abstract
Bsp.: Hotelseite mit Unterseiten für Ausstattung, Zimmer und Preise

m:n mehrere Dokumente → mehrere abstracts
Bsp.: Übersichtsseite mit Unterseiten für Adresse und Preise pro angegebenes Hotel

Aufgrund dieser Abbildungen gibt es verschiedene Möglichkeiten des Zielstrukturaufbaus. Die nachfolgend vorgeschlagene ist eine dieser Möglichkeiten.

Zielstruktur Die Zielstruktur wird pro Server bzw. bei einzelnen Seiten pro Seite durch ein XML–Dokument ausgedrückt, welches in zwei wesentliche Teile aufgeteilt ist. Im ersten Teil werden die Dokumente mit ihren Metadaten und ihren Beziehungen untereinander beschrieben. Im zweiten Teil folgen die abstracts mit dem Verweis auf eine Menge von Dokumenten, aus denen das jeweilige abstract entstanden ist.
Die Zielstruktur wird folgende Informationen enthalten:

1. Daten zum jeweiligen Dokument:
   - URL — als Identifikator des Dokumentes
   - Links — Menge von URLs zu verlinkten Dokumenten

- Klassen — Menge von zugehörigen domänenspezifischen Klassen und eine funktionale Klasse (DOMAIN-CLASSES, FUNCTION-CLASS)
- keywords — Menge von keywords, domänenspezifische sowie Information Retrieval–keywords sowie angegebene (Metatags bei HTML)
- Metadaten — Update-Time, Refresh-Time, Delete-Time, eventuell weitere

2. Daten zum jeweiligen abstract:

- abstract — das eigentliche abstract als Baum
- URL–Menge — Menge der URLs der Dokumente, die für die Bildung des abstracts verantwortlich sind

Aufbau der Zielstruktur Die URL und die Links werden durch den Gatherer gewonnen und unmittelbar nach dem Sammel– und Summarizing–Prozeß an die Zielstruktur übergeben. Die Klasse wird im Klassifizierungs–Tool im Datenaufbereitungsprozeß bestimmt. Die keywords werden je nach Analyseart zu unterschiedlichen Zeitpunkten in der Zielstruktur ergänzt. Die meisten Metadaten werden beim Summarizing bestimmt und ebenfalls der Zielstruktur übergeben.
Das abstract wird im letzten Datenaufbereitungsschritt aufgebaut. Die URL–Verweise zu den ursprünglichen Originaldokumenten werden bei der Kombination der abstracts gesammelt und zum Ende dieses Prozesses in die Zielstruktur geschrieben.

Beispiel einer Zielstruktur Die im Anhang A beschriebene HTML–Seite wird hier nun die Grundlage der Zielstruktur bilden. Dabei werden die wichtigsten Daten in einem XML–konformen Beispielausschnitt dargestellt.

```
<WWW--SITE>
    <URL>www.all-in-all.de/1127.htm</URL>
    <LINKS>
        <LINK>www.all-in-all.de/...htm</LINK>
    </LINKS>
    <DOMAIN-CLASSES>unterkunft</DOMAIN-CLASSES>
    <FUNCTION-CLASS>major concept site</FUCTION-CLASS>
    <IR-KEYWORDS>
        Hotel, Zimmer, TV, Radio
    </IR-KEYWORDS>
    <META-DATA>
        <TYPE>HTML</TYPE>
        <MD5>dsf34srrt45hetr8454hu5hg3h4</MD5>
        <UPDATE>25.06.2000, 20:35:25</UPDATE>
        <REFRESH>14d</REFRESH>
        <DELETE>2m</DELETE>
    </META-DATA>
</WWW-SITE>
<ABSTRACT>
    <UNTERKUNFT>
        <HOTEL>atrium hotel krueger
            <ORT>Rostock</ORT>
            <ZIMMER>einzelzimer
                <PREIS>138
                    <PREIS_WAEHRUNG>dm</PREIS_WAEHRUNG>
                </PREIS>
            </ZIMMER>
            <ZIMMER>doppelzimmer
                <PREIS>125
```

```
                    <PREIS_WAEHRUNG>dm</PREIS_WAEHRUNG>
                </PREIS>
            </ZIMMER>
        </HOTEL>
    </UNTERKUNFT>
    <URLS>
        <URL>www.all-in-all.de/1127.htm</URL>
    </URLS>
</ABSTRACT>
```

4.5 Zusammenfassung

In diesem Kapitel wurde beschrieben, wie eine Lösung des motivierten Informationsextraktionsproblems aussehen kann. Die herausgestellte Architektur besteht im wesentlichen aus zwei Hauptkomponenten. Zum einen ist es das eigentliche Gathering, wie es aus herkömmlichen Internetsuchmaschinen bekannt ist. Zum anderen ist es die Datenanalyse und -aufbereitung. Bei der Datenanalyse werden durch drei verschiedenartige Analyse–Tools die gesammelten Dokumente auf Struktur, Linguistik und Domänenspezifika überprüft. Durch weiterführende Datenverarbeitungsschritte wird eine Zielstruktur für die zusammenhängenden Dokumente und die daraus resultierenden abstracts aufgebaut. Die XML–Zielstruktur wird dann der Speicherkomponente übergeben.
Es wurden Lösungen für das Klassifizierungsproblem, das Kombinationsproblem (siehe 1.3.2) und vor allem für die Verbesserung der momentanen abstract–Generierung in GETESS (Kapitel 3.3) gezeigt.

Bevor eine konkrete Realisierung und Implementierungsvorschläge zur Informationsextraktion beschrieben werden, betrachtet dieser Abschnitt einige Voraussetzungen und Einschränkungen für die vorgestellten Konzepte der Informationsextraktion.

- Vorliegen eines domänenkonformen Textdokuments mit keinem bis starkem Strukturanteil.
- Konzepte aus Domänenlexikon, Ontologie und den konkreten Daten in der Datenbank müssen zueinander abbildbar sein.
- Aufgrund möglicher Überlappungen zwischen der Semantik der verschiedenen Heuristiken in der Strukturanalyse, in der linguistischen Analyse und in der domänenspezifischen Analyse sollten die angewendeten Heuristiken zur Informationsextraktion zueinander abgestimmt sein.
- Domänenspezifische Speziallexika für konkrete Daten, die wichtig für die Informationsextraktion sind, werden benötigt.
- Die Ontologiekonzepte und -relationen müssen bestimmten Voraussetzungen genügen. Zum einen wird eine Hierarchie von “part of”– und “is a”–Beziehungen zur Klassifizierung der Dokumente und Dokumentteile vorausgesetzt. Zum anderen benötigt man eine Ontologie mit eindeutigen Relationen zur abstract–Generierung. Die Ontologie bildet den Rahmen bzw. das Template für die abstracts.

Kapitel 5

Realisierung im GETESS–Gatherer

Kapitel 4 hat gezeigt, wie eine Informationsextraktion mit Hilfe von strukturellen und inhaltlichen Analyseschritten aufgebaut werden kann. In diesem Kapitel wird erläutert, wie die Konzeption für den konkreten Einsatz im Projekt GETESS umgesetzt werden kann.
Es wird gemäß des Architekturvorschlags in Kapitel 4 die Umsetzung der Teilkomponenten nachfolgend beschrieben. Anschließend wird die Funktionalität der Informationsextraktion an einem Beispiel dargestellt.
Zum Ende des Kapitels wird eine Bewertung und Evaluierung des Ansatzes mit weiteren Beispielen vorgenommen.

5.1 Architektur der Informationsextraktion in GETESS

Ausgangspunkt für die nun folgende konkrete Architektur für die GETESS–Informationsextraktion ist die Architekturskizze 4.1 aus Kapitel 4.
Die Architekturabbildung 5.1 zeigt das GETESS–Gathering und die Datenanalyse und -aufbereitung.

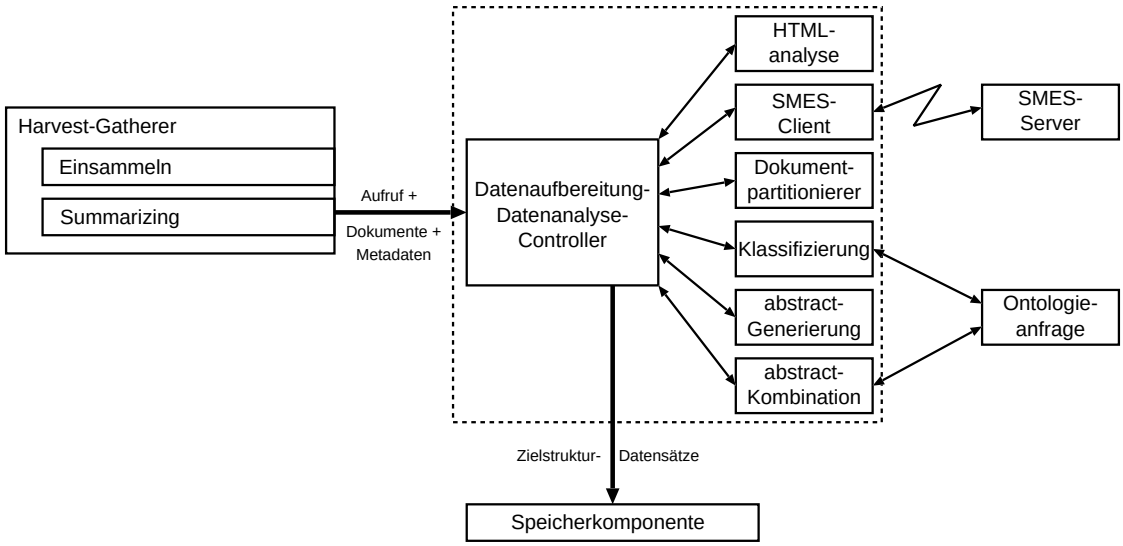

Abbildung 5.1: Architekturskizze der Informationsextraktion in GETESS

Das Gathering wird von Harvest durchgeführt. Wobei Harvest nach Angabe der zu durchsu-

chenden Seiten den Sammelagenten startet und mittels Summarizing Metadaten extrahiert. Vom Summarizing aus erfolgen der Aufruf des Controllers zur Datenanalyse und -aufbereitung und die Übergabe der Dokumente und Metadaten.

Bei der Datenanalyse wird zunächst die HTML–Analyse und die linguistische, domänenspezifische Analyse mit SMES durchgeführt. Die HTML–Analyse gibt eine HTML–Baumstruktur zurück. SMES liefert linguistisch ermittelte Phrasen mit Bezug auf Ontologiekonzepte. Diese Phrasen stellen die domänenspezifischen keywords dar. Anschließend werden die ermittelten Strukturdaten und die linguistischen, domänenspezifischen Daten kombiniert.
Die folgenden Datenverarbeitungsschritte benötigen die ermittelten Analyseergebnisse. Dazu gehört neben der Dokumentpartitionierung die Klassifizierung, die einmal eine funktionelle Klasse und zum anderen domänenspezifische Klassen pro Dokument bestimmt. Zur Bestimmung der domänenspezifischen Klassen sind die Daten aus der Ontologie nötig.
Die abstracts werden ähnlich wie bei der Klassifizierung mittels Ontologiewissen gebildet und kombiniert. Die fertige Zielstruktur wird der Speicherkomponente übergeben.

Die einzelnen Komponenten werden nun in den nächsten Abschnitten genauer betrachtet.

5.2 Umsetzung der Gathering–Komponente

Die Gathering–Komponente in GETESS wird durch eine Installation des Harvest–Gatherers gestellt. Harvest [HSW96] ist eine Internet–Suchmaschine mit weitreichenden Tools zum Gathering, zur Datenaufbereitung, zum Speichern und zum Nutzerdialog. Harvest wurde von der Internet Research Group on Resource Discovery (kurz IRTF-RD) entwickelt. Von der Universität von Edinburgh wurde Harvest noch bis zur Version 1.5.20 Ende 1996 weiterentwickelt. Danach sind keine wesentlichen Entwicklungsschritte mehr vollzogen worden. Laut [BYRN99] sind allerdings immer noch sehr viele Installationen von Harvest im Internet zu finden.
Die Vorteile von Harvest liegen in der möglichen verteilten Installation und in der Möglichkeit, die Software auf einfache Weise zu erweitern.
Für die angestrebte Informationsextraktion in GETESS ist die Gathering–Komponente von Harvest vordergründig interessant.

5.2.1 Der Harvest–Gatherer

Der Harvest–Gatherer besteht im wesentlichen aus einer Sammelvorbereitung, dem eigentlichen Sammelprozeß und der Datenaufbereitung (Summarizing).
Bei der Sammelvorbereitung geht es neben dem Bereitstellen der zu durchsuchenden Serveradressen auch um ein mögliches Update[1] vorangegangener Gathering–Läufe. Das Eintragen der zu durchsuchenden Adressen wird in der Gatherer–Konfigurationsdatei durchgeführt. Es gibt dabei eine Sektion für die kompletten Server und eine Sektion für einzelne Seiten. Zusätzliche Angaben, wie Suchtiefe oder Access–Methode (siehe dazu Harvest–Manual [HSW96]), können an dieser Stelle ebenfalls angegeben werden.
Beim Sammelprozeß wird der Harvest–Sammelagent mit Hilfe einer übergebenen URL das jeweilige Dokument einsammeln. Der für die Informationsextraktion interessanteste Teil ist die Datenaufbereitung mit dem Summarizing.

Die Datenaufbereitung ist Dokumenttyp–abhängig. Der erste Schritt ist die Ermittlung des Dokumenttyps. Für jeden unterstützten Dokumenttyp gibt es ein Summarizer–Programm.
Bei der Datenaufbereitung geht es vor allem um das Extrahieren der Link–Informationen als Eingabe für die Sammelagenten und um das Extrahieren und Zusammenstellen der Metadaten und keywords für die Harvest–Indexierung. Die generierten Daten des Summarizing werden in sogenannten SOIF–Datensätzen (Summary Object Interchange Format) gesammelt und der Indexierung übergeben.

[1] Der Aspekt des Updates der Daten wurde bisher nicht betrachtet. Auf die wesentliche Problematik dieser wichtigen Funktion einer Suchmaschine wird im Ausblick noch kurz genauer eingegangen.

5.2.2 Anbindung des Datenanalyse– und -aufbereitungs–Controller

Der Aufruf der GETESS–Informationsextraktion vom Gathering aus geschieht innerhalb des Summarizing. Dabei werden Dokument und Metadaten übergeben. Zu den Metadaten gehört in erster Linie die URL als Identifikator des Dokuments.

Die meisten Summarizer–Programme sind Script–Programme (Shell, Perl etc.), so daß es relativ einfach ist, weitere Programme in den Prozeß einzubinden. In GETESS wird sich zum jetzigen Zeitpunkt auf HTML–Dokumente beschränkt, da es keine weiteren textuellen Internet–Dokumenttypen innerhalb der Domäne "Tourismus in Mecklenburg/Vorpommern" gibt.
Innerhalb des HTML–Summarizers wird nun der Datenanalyse–Controller aufgerufen und die URL und das Dokument (als Dateiname) als Parameter übergeben.

5.3 Umsetzung der Datenanalyse und -aufbereitung

In diesem Abschnitt werden die Controller–Komponente, die Analyse–Programme und die Datenaufbereitung mit dem Resultat der abstract–Datensätze betrachtet.

5.3.1 Der Controller

Der Harvest–Summarizer ruft den Controller auf und übergibt ihm das Dokument und die URL (weitere Metadaten können auch übergeben werden). Im Controller werden nun nacheinander zuerst die Analysetools, dann die Klassifizierung, die Dokumentunterteilung und die abstract–Generierung und –Kombination aufgerufen. Zuletzt werden die resultierenden Datensätze an die Speicherkomponente übergeben.
Der Controller besteht aus einem Perl–Script, in dem flexibel weitere Werkzeuge an beliebiger Stelle eingesetzt und auch wieder entfernt werden können. Der Controller legt das Arbeitsverzeichnis fest und kann je nach Bedarf Dokument, URL oder Ergebnis–Files an die Werkzeuge übergeben.

5.3.2 Analysewerkzeuge

Im GETESS–Informationsextraktionsprozeß werden zwei Analyseprogramme unterstützt. Zum einen ist es die HTML–Strukturanalyse und zum anderen die linguistische, domänenspezifische Analyse mit SMES (SMES beinhaltet somit zwei Analyseschritte).

HTML–Strukturanalyse

Das Resultat der HTML–Strukturanalyse ist ein HTML–Baum nach OEM–Modellierung (siehe 2.3). Dazu werden zwei Perl–Bibliotheken, der HTML-Parser von [AC00] und das HTML-Tree–Modul von [AB00], verwendet. Die Parser–Bibliothek hat als Kernstück ein in der Programmiersprache C geschriebenes Parsing–Programm, welches eine ansprechende Performance der Strukturanalyse verspricht.
Das Ergebnis, der HTML–Baum, wird in einer dem Dokument zugehörigen Strukturdatei gespeichert.

Linguistische, domänenspezifische Analyse

Die linguistische, domänenspezifische Analyse wird im GETESS–Projekt durch das SMES–System durchgeführt. Der hier verfolgte Ansatz zur Anbindung von SMES in den Gathering–Prozeß ist sehr ähnlich der in [Bru00] beschriebenen Vorgehensweise. Dabei wird der SMES–Client lokal beim Gatherer installiert und die Parameter des SMES–Servers angepaßt. Der Controller ruft mit dem folgenden Ausdruck den SMES–Client auf und übergibt ihm neben den SMES–Parametern das Dokument.

```
parac -m smes::fst-from-string ((:fst . :all-frags)) -t text/xml -i 1127.htm
>1127.htm.ling
```

Das Ergebnis mit den linguistischen und domänenspezifischen Informationen wird in einem File abgelegt.
Die linguistischen Informationen sind die in Kapitel 3 gezeigten Phrasen der SMES–Analyse. Den

einzelnen Phrasen werden domänenspezifische Konzepte aus dem Domänenlexikon zugeordnet.

Mit dieser Analyse sind linguistisch generierte, domänenspezifische keywords entstanden. Diese keywords bilden die ersten inhaltsbasierten Daten. Diese bedürfen allerdings noch einer weiteren Bearbeitung.

5.3.3 Datenverarbeitungs–Komponenten

Dokumentaufteilung

Die Dokumentaufteilung ist ein Produkt der Kombination von strukturellen Daten und den ermittelten, domänenspezifischen keywords. Dazu werden die keywords den entsprechenden Teilstrukturen des HTML–Baums zugeordnet. Mittels der Heuristiken aus 4.3.5 werden die Teilstrukturen bestimmt. Teilstrukturen, denen keine keywords zugeordnet werden konnten, werden in den folgenden Verarbeitungsschritten nicht mehr verwendet.

Die Umsetzung erfolgt mittels mehrerer kleiner Perl–Programme. Dabei wird zuerst die Zuordnung zwischen den keywords und dem Strukturbaum hergestellt. Das Problem besteht darin, daß die keywords eine linguistisch motivierte Positionierung der keywords innerhalb des Dokuments besitzen, die nicht direkt auf die Ortskoordinaten des HTML–Baumes abbildbar sind. Zur indirekten Abbildung ist das Originaldokument nötig.
Mit der Anpassung der Heuristiken an das konkrete Anwendungsszenario werden die einzelnen Dokumentteile bestimmt. Die Anpassung geschieht einmal bei der Größe der einzelnen Teile und zum anderen bei den möglichen Aufspaltungspunkten. Konkret wird die Anzahl der Konzepte pro Teilstück momentan auf drei bis fünf angestrebt. Die Aufspaltungspunkte entsprechen den in 4.3.5 vorgeschlagenen strukturellen HTML–Tags.

Klassifizierungs-Tool

Wie in der Konzeption in Kapitel 4 dargelegt, besteht die vorgeschlagene Klassifizierung der Dokumente aus der Bestimmung einer funktionellen Klasse und der Bestimmung einer oder mehrerer domänenspezifischer Klassen.

Bestimmung der domänenspezifischen Klassen Zur Bestimmung der domänenspezifischen Klassen werden die keywords des jeweiligen Dokuments und die Ontologiedaten benötigt. Die Ontologiedaten werden aus der GETESS–Ontologiedatenbank (siehe [MPP+99]) (auf IBM DB2 Universal Database Version 6) gewonnen. Aufgrund der vielen rekursiven Zugriffe pro Dokument beim Verfolgen von Referenzen werden die Ontologiedaten einmal für den gesamten Gathering–Prozeß aus der Datenbank geholt und für den benötigten Zugriff optimiert abgelegt. Dabei wird die von der Ontologie abweichende Darstellung der Konzepthierarchie aus der tatsächlichen Speicherstruktur in der Datenbank benutzt. Abweichend ist, daß abstrakte Konzepte und sehr konkrete Konzepte aus der Hierarchie herausgenommen sind, wobei die sehr konkreten Konzepte als Attribut des übergeordneten Konzept wiederzufinden sind.
Aus diesem Schritt ergeben sich dann zwei Hierarchien. In der ersten befinden sich die Ontologiekonzepte mit ihrer "is a"–Hierarchie. In der zweiten sind alle möglichen, direkten Referenzen ("part of"–Beziehungen) der Konzepte zu finden.
Aufgrund des Anwendungsszenarios ergibt sich eine sinnvolle Vereinfachung und Verbesserung der Klassenfindung, indem die Suche nach einer Einordnung in die Klassenhierarchie auf den Titel der HTML–Dokumente beschränkt wird. Sollte der Titel nicht ausreichen oder nicht vorhanden sein, besteht weiterhin die Möglichkeit daß gesamte Dokument zur Klassenfindung zu nutzen.

Mit den eben genannten Möglichkeiten wird der Algorithmus aus Kapitel 4.3 zur Bestimmung der domänenspezifischen Klasse abgearbeitet. Das Ergebnis ist damit eine Menge von domänenspezifischen Konzepten.

Bestimmung der funktionellen Klasse Die funktionelle Klasse wird mittels der Teilstrukturen, der domänenspezifischen keywords und der domänenspezifischen Klassenbestimmung ermittelt.

Für jede Teilstruktur wird das keyword mit dem jeweiligen am höchsten stehenden Konzept in der Konzepthierarchie ausgewählt. Diese keywords bilden dann eine Menge, und aufgrund ihrer Beziehungen und dem direkten Vergleich der Instanzen der Konzepte wird die funktionelle Klasse, wie in 4.3 beschrieben, entschieden. Auch hierbei können bestimmte, strukturelle Abschnitte, wie der Titel oder Überschriften helfen.

5.3.4 abstract–Generierung

Als abschließende Datenverarbeitung startet der Controllor nacheinander die Generierung der Teil–abstracts und die Kombination der abstracts.

Generierung der Teil–abstracts Zum Generieren der Teil–abstracts werden die in der Dokumentseparation ermittelten kleinen Strukturen an die in Kapitel 3 gezeigte SMES–Analyse übergeben. Bei der SMES–Analyse werden nun diese kleinen Dokumentstrukturen linguistisch geparst, auf Domänenwissen abgebildet und mittels Ontologieanfragen Relationen über die ermittelten Konzepte gebildet.
Der SMES–Client–Aufruf lautet folgendermaßen:

```
parac -m smes::pairs-from-html ((:fst . :all-frags)) -t text/xml -i
1127.htm.1 >1127.htm.1.abstr
```

Als Ergebnis werden die Relationen in einem XML–File pro Teilabstract abgelegt. Die Relationen sind aus der Ontologie durch linguistisch motivierte Heuristiken entstanden.

Kombination der Teil–abstracts Zur Kombination der Teil–abstracts benötigt man Ontologiewissen.
Die erste Kombination wird innerhalb des Dokuments durchgeführt. Im zweiten Schritt werden die Dokumente in der mit Links verbundenen Umgebung auf Kombinationsmöglichkeiten hin untersucht.

Der Ausgangspunkt zur Kombination ist das Teil–abstract (eventuell auch mehrere), in dem die jeweilige domänenspezifische Dokumentklasse gefunden wurde. Die Dokumentklasse ist damit der Einstiegspunkt für das Finden von Relationen in der Ontologie. Von diesem Einstiegspunkt aus werden zuerst die Relationen des Einstiegs–abstracts und dann die der weiteren Teil–abstracts überprüft und gegebenfalls durch weitere Relationierungen angelagert.
Bei der Kombination werden zuerst die Instanzen und Terme gleicher Konzepte verglichen. Danach werden Beziehungen zwischen den Konzepten aufgrund des Ontologiewissens gebildet, wobei die Ontologiehierarchie mit den Relationen ausgehend von der Dokumentklasse als Wurzel aller Beziehungen verfolgt wird.
Das resultierende abstract besteht dann aus einer Menge von folgenden Relationen:

- "IS_A" — aus der Ontologiehierarchie ergebene Relation.
- "HAS_RELATION_WITH" — Gerichtete Beziehung zwischen Konzepten basierend auf Ontologierelationen.
- "EQUAL" — entsteht beim Relationieren von Termen die zu einem gemeinsamen Konzept zusammengehören.

Zwei weitere Relationen "SAME_FATHER" (gemeinsamer Vaterknoten) und "SAME_ROOT" (gleiche Wurzel) werden auf die vorher genannten Beziehungen aufgelöst oder als nicht relationierbar nicht weiter betrachtet. Die resultierenden abstracts werden zusammen mit den Daten der Dokumente in das Ergebnisfile gespeichert.

Die angestrebte Zielstruktur im GETESS–Anwendungsszenario weicht in der Darstellung von der konzeptionellen Zielstruktur in 4.4 ab. Diese Abweichungen resultieren aus Vorgaben der momentanen Schnittstelle zwischen Gathering und Speicherkomponente bzw. entstehen aufgrund von fehlenden Informationen aus der linguistischen, domänenspezifischen Analyse. Eine Erweiterung der Schnittstelle und eine Verbesserung der Analyseergebnisse wird angestrebt.

5.4 Beispiel der Informationsextraktion in GETESS

Ausgangspunkt für dieses Beispiel der Informationsextraktion in GETESS sind die hier vorgestellten implementierten Tools und die konkrete HTML–Seite aus dem Anhang A.
Das Dokument im Anhang A ist eine Hotelbeschreibung vom Internet–Server "www.all-in-all.de". Diese Seite wird zunächst dem Gatherer durch einen Eintrag in die Liste der zu durchsuchenden Seiten bekanntgemacht. Daraufhin wird der Gathering–Prozeß gestartet und die Seite durch den Agenten eingesammelt.
Wenn der Harvest–Gatherer den HTML–Summarizer zum Information Retrieval der Gathering–Daten anspricht, wird der Informationsextraktions–Controller gestartet. Hier wird zuerst die HTML–Struktur analysiert. Daraus resultiert dann ein HTML–Baum, der folgendermaßen aussieht (Ausschnitt, kompletter Baum im Anhang A):

```
<node pos='0' tagname='html'/>
<node pos='0.0' tagname='head'/>
<node pos='0.0.0' tagname='title'/>
<node pos='0.1' tagname='body'/>
<node pos='0.1.0' tagname='div'/>
<node pos='0.1.0.0' tagname='center'/>
<node pos='0.1.0.0.0' tagname='table'/>
<node pos='0.1.0.0.0.0' tagname='tr'/>
<node pos='0.1.0.0.0.0.0' tagname='td'/>
<node pos='0.1.0.0.0.0.0.0' tagname='p'/>
<node pos='0.1.0.0.0.0.0.0.0' tagname='a'/>
<node pos='0.1.0.0.0.0.0.0.0' tagname='img'/>
<node pos='0.1.0.0.0.0.1' tagname='td'/>
<node pos='0.1.0.0.0.0.1.0' tagname='p'/>
<node pos='0.1.0.0.0.0.1.0.1' tagname='a'/>
<node pos='0.1.0.0.0.0.1.0.3' tagname='a'/>
<node pos='0.1.1' tagname='div'/>
```

Es werden hier alle Tags außer "META" und Programmierschnittstellen angeordnet.
Der nächste vom Controller gestartete Analyseschritt ist die domänenspezifische, linguistische keyword–Extraktion. Dazu wird der SMES–Client mit den dafür spezifizierten Parametern aufgerufen. Als Ergebnis erhält man eine Liste von erkannten linguistischen Strukturen. Nachdem die mit domänenspezifischen Konzepten angereicherten keywords entnommen sind, werden diese keywords in die Baumstruktur eingeordnet.
Der folgende Auszug zeigt diese keywords mit Konzept und Positionsangabe.

```
<HEAD>hotel</HEAD>
<DOMAIN-TYPE>Hotel</DOMAIN-TYPE><POS>0.0.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.0.0</POS>
<HEAD>bad</HEAD>
<DOMAIN-TYPE>Bad</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.6.1.0</POS>
<HEAD>telefon</HEAD>
<DOMAIN-TYPE>Telefon</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.6.1.0</POS>
<HEAD>tv</HEAD>
<DOMAIN-TYPE>Fernseher</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.6.1.0</POS>
<HEAD>radio</HEAD>
<DOMAIN-TYPE>Radio</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.8</POS>
<HEAD>minibar</HEAD>
<DOMAIN-TYPE>Minibar</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.8</POS>
<HEAD>einzelzimmer</HEAD>
<DOMAIN-TYPE>Einbettzimmer</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.11.1</POS>
<HEAD>doppelzimmer</HEAD>
```

```
<DOMAIN-TYPE>Zweibettzimmer</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.11.1</POS>
<HEAD>dreibettzimmer</HEAD>
<DOMAIN-TYPE>Mehrbettzimmer</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.11.3</POS>
```

Damit sind die Analyseschritte beendet und der Controller startet die Datenverarbeitungs–Tools.
Im ersten Datenverarbeitungsschritt wird die Ontologieklasse bestimmt. Das Dokument hat einen Titel, somit kann eine vereinfachte Klassenfindung durchgeführt werden. Beim vorliegenden Dokument stehen vier Konzepte innerhalb der Titelumgebung zur Auswahl, "Urlaub", "Hotel", "Region" und "Stadt". Nach der verwendeten Ontologie ergibt sich "Hotel" als übergeordnete Klasse. Eine weitere wichtige linguistische Information ist, daß es sich hier im Titel nur um ein Hotel handelt. Es handelt sich demnach wahrscheinlich um eine Hauptkonzeptseite. Die Analyse der Teil–abstracts bestätigt diese Annahme.
Der nächste Datenverarbeitungsschritt ist die Separation von Dokumentteilen. Das vorliegende Dokument wurde in 4 Teile zerlegt. In diesen Teilen wurden von der linguistischen Analyse meist zwischen 2 und 6 (es können auch mehr sein, wenn keine weitere Aufspaltung möglich war) domänenspezifische keywords gefunden.
Die folgenden Ausschnitte zeigen zwei der generierten Dokumentteile.
Der erste Ausschnitt beschreibt den Titel.

```
<head><title>
Das Atrium Hotel Kr&uuml;ger in Rostock - Urlaub und Reisen in
Mecklenburg
</title></head>
```

Der nächste Ausschnitt ist der dritte aus der Reihe der Dokumentteile und beschreibt die Lage sowie die Preise.

```
Lage:
Das von der Familie Kr&uuml;ger gef&uuml;hrte GARNI-HOTEL im Westen
der Hansestadt Rostock pr&auml;sentiert sich als ein modernes
Gesch&auml;fts- und Messehotel. Touristen oder Gesch&auml;ftsreisende
sch&auml;tzen die angenehme und freundliche Atmosph&auml;re.
Hausbeschreibung:
Die 59 Komfortzimmer und Appartements sind mit Bad oder Dusche/WC,
Telefon, TV, Radio, Minibar und F&ouml;n ausgestattet.
Preise: (inclusive Fr&uuml;hst&uuml;ck)
Einzelzimmer: von 95,00 - 125,00 DM
Doppelzimmer: 150,00 DM,
Dreibettzimmer: 185,00 DM
Appartements: von 135,00 - 190,00 DM
```

Im Originaldokument sind "Lage:" und "Preise:" fett hervorgehoben, der komplette Text hat aber einen gemeinsamen direkten Vaterknoten im Hierarchiebaum. Hervorgehobene Textstellen, wie Fettschreiben oder Unterstrichen werden im allgemeinen für kurze Textstellen verwendet und sind für die Strukturanalyse wenig ausagekräftig.

Die einzelnen Ausschnitte werden nun durch den SMES–Client zur abstract–Generierung an die linguistische, domänenspezifische Analyse geschickt. Das Ergebnis ist eine Menge von Teil–abstracts.
Das folgende Teil–abstract ist ein Ausschnitt des SMES–Ergebnis aus dem ersten Dokumentteil, dem Titelteil.

```
<tuple name='getess-output'>
  <fragment>
    <type>Stadt</type><inst>rostock</inst><name>Rostock</name>
  </fragment>
  <fragment>
```

```
    <type>Region</type><inst>mecklenburg</inst><name>Mecklenburg</name>
  </fragment>
</tuple>
<tuple name='getess-output'>
  <fragment>
    <type>Stadt</type><inst>rostock</inst><name>Rostock</name>
  </fragment>
  <fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst><name>urlaub</name>
  </fragment>
</tuple>
</tuple>
<tuple name='getess-output'>
  <fragment>
    <type>Hotel</type><inst>Hotel_1</inst><name>hotel</name>
  </fragment>
  <fragment>
    <type>Hotel</type><inst>Hotel_1</inst><name>krueger</name>
  </fragment>
</tuple>
<tuple name='getess-output' rel='liegt_in_Stadt'>
  <fragment>
    <type>Hotel</type><inst>Hotel_1</inst><name>hotel</name>
  </fragment>
  <fragment>
    <type>Stadt</type><inst>rostock</inst><name>Rostock</name>
  </fragment>
</tuple>
```

Ausgehend von der vorher bestimmten domänenspezifischen Dokumentklasse wird nun zuerst innerhalb des Titel–abstracts, danach in allen weiteren Teil–abstracts nach zugehörigen Relationen gesucht.
Aus dem oben dargestellten Teil–abstract ergeben sich beispielsweise folgende Relationen:

```
<tuple name='getess-output' rel='EQUAL'>
  <fragment>
    <type>Hotel</type><inst>Hotel_1</inst><name>hotel</name>
  </fragment>
  <fragment>
    <type>Hotel</type><inst>Hotel_1</inst><name>krueger</name>
  </fragment>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'>
  <fragment>
    <type>Hotel</type><inst>Hotel_1</inst><name>hotel</name>
  </fragment>
  <fragment>
    <type>Stadt</type><inst>rostock</inst><name>Rostock</name>
  </fragment>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'>
  <fragment>
    <type>Hotel</type><inst>HOTEL_1</inst><name>hotel</name>
  </fragment>
  <fragment>
```

```
      <type>Urlaub</type><inst>Urlaub_1</inst><name>urlaub</name>
    </fragment>
  </tuple>
  <tuple name='getess-output' rel='HAS_RELATION_WITH'>
    <fragment>
      <type>Hotel</type><inst>Hotel_1</inst><name>hotel</name>
    </fragment>
    <fragment>
      <type>Region</type><inst>mecklenburg</inst><name>Mecklenburg</name>
    </fragment>
  </tuple>
```

Die im Teil–abstract relationierten Konzepte Stadt und Region haben die "SAME_FATHER"–Relation. Diese Relation wird nicht in das Ziel–abstract aufgenommen. Wenn allerdings der gemeinsame Vater mit einem Konzept ausgehend von der Hauptklasse Hotel relationierbar ist, werden beide über diese Relation aufgenommen. Die kompletten Teil–abstracts und das Ziel–abstract sind im Anhang C zu finden.

5.5 Bewertung und Beispiele der realisierten Lösungen

Kapitel 4 und das bisherige Kapitel 5 haben gezeigt, daß durch komplexe Analyse- und Verarbeitungsprozesse eine erfolgversprechende Informationsextraktion möglich ist. Die Bewertung und Evaluierung im vorliegenden Abschnitt werden versuchen, die Machbarkeit und Relevanz der vorgeschlagenen Konzepte und Realisierungen zu unterlegen.
Das Erreichen des Hauptziels dieser Arbeit, die Verbesserung der bisherigen abstracts in GETESS, wird durch einige Beispiele und generelle Betrachtungen gezeigt.

5.5.1 Generelle Aspekte

Im folgenden werden einige Einschränkungen und weiterführende Gedanken zur Realisierung im GETESS–Gatherer genannt.

- Derzeit liegt eine Beschränkung auf HTML–Dateien vor, da momentan die meisten Informationen in dieser Form im WWW dargestellt werden.

- Die Strukturanalyse wird auf einer normalisierten HTML–Seite durchgeführt. Die Normalisierung besteht in einer Strukturumwandlung der HTML–Seite, wobei bestimmte, nicht strukturelle Bestandteile, wie Formular– und Programmier–Tags, nicht übernommen werden.

- Die Ontologie bietet momentan nur die Unterscheidung zwischen "is a"–Beziehungen und weiteren Beziehungen aller Art, wobei es Mehrdeutigkeiten geben kann. Diese Ontologie kann nur eingeschränkt eingesetzt werden. Für die Relationierung von Konzepten wären Kardinalitäten an den Ontologierelationen sehr hilfreich.

- Die linguistische Analyse bietet noch nicht alle benötigten Spezialgrammatiken, so daß beispielsweise die Adresserkennung noch unzureichend ist. Weiterhin wäre eine Aussage über den Numerus (Singular/Plural) von Termen interessant.

- Um die Daten in das bestehende Datenbanksystem zu integrieren, wird die Schnittstelle zur Datenbank zu überarbeiten sein. Es wird deshalb nur eine eingeschränkte Version der Zielstruktur momentan implementiert.

- Die Schnittstelle zwischen der SMES–Analyse und dem Gathering–Prozeß ist ebenfalls unzulänglich, da das Zuordnen der gewonnenen domänenspezifischen Konzepte auf die konkrete Instanz in der HTML–Seite momentan sehr umständlich auf Gathering–Seite gemacht werden muß.

- Die Performance beim Gathering ist momentan von der Anzahl der nötigen Internetverbindungen abhängig ([Bru00]). Dadurch entsteht eine gewisse Verschlechterung der Gathering–Zeiten, da mit dieser Konzeption zwei weitere Internet–Verbindungen dazukommen. Wenn daß System als Ganzes innerhalb einer Umgebung ohne Internet–Antwortzeiten[2] operiert, wird die Performance wahrscheinlich um einiges besser sein.

- Die Performance des Kombinierens hat sich aufgrund umfangreicher Relationierungsbemühungen bei einer sehr großen Ontologie als problematisch herausgestellt. Hierbei sind allerdings weitgehende Optimierungen denkbar.

5.5.2 Beispiele zur Evaluierung des Ansatzes

In Abschnitt 5.4 wurde ein Beispiel des Internet–Servers "www.all-in-all.de" betrachtet. Dabei handelte es sich um eine Hotelbeschreibung des Informationsanbieters auf dessen Internet–Server. Zunächst wird das Ergebnis des Beispiels aus Abschnitt 5.4 mit dem früheren GETESS–Ergebnis (ohne Anwendung der hier vorgestellten Konzepte) aus Kapitel 3.3 bzw. Anhang B an einigen Aspekten verglichen.

Vergleich der bisherigen mit der aktuellen Analyse Aufgrund der eingeführten Lokalität der Informationen durch Separation des Dokuments in Dokumentteile, werden im Gegensatz zum früheren Ansatz nicht mehr Relationierungen über Strukturgrenzen hinaus, sondern im ersten Schritt nur für lokale Strukturen bestimmt.
Relationierungen wie die folgende können mit den hier vorgestellten Mechanismen nicht mehr auftreten.

```
<tuple name='getess-output'>
  <fragment>
    <type>Hotel</type><inst>hotel_1</inst><name>hotel</name>
  </fragment>
  <fragment>
    <type>Stadt</type><inst>schwerin</inst><name>Schwerin</name>
  </fragment>
</tuple>
```

Schwerin war hier lediglich der Sitz des Informationsanbieters und stand ganz unten auf der HTML–Seite.
Durch die Auswertung des konkreten Typs der Ontologierelationen werden auch die vielen Überkreuz–Relationierungen, wie bei Radio, Telefon, TV und Fön, vermieden. Diese Konzepte werden nun durch die Beziehung des Vaterkonzeptes "Austattung" mit den Zimmern relationiert.
Ein wichtiger Punkt ist auch die Verwendung der Dokumentklasse als Einstiegspunkt in die Ontologie. Dadurch wird vermieden, daß die Konzepte wahllos untereinander in Beziehung gesetzt werden, ohne eine korrekt durchgehende Struktur zu erhalten.
Der im Beispiel 3.3 häufig aufgetretenen, mehrmaligen Instanziierung ein und desselben Konzeptes wird einmal durch die Separierung begegnet, und zum anderen können durch die Klassenbestimmung des Dokuments und Kardinalitäten an den Ontologierelationen die Instanzen überprüft werden.

Weitere Beispiele Zwei weitere Beispiele von Dokumenten aus der Tourismusbranche in Mecklenburg/Vorpommern sind im Anhang D dargestellt. Es handelt sich dabei einerseits um eine anders strukturierte Aufmachung von Hotelinformationen und anderseits um eine andere Funktionalität innerhalb eines Tourismusservers.
Das erste Dokument ist eine Darstellung zu Zimmer und Suiten eines Hotels mit einer eigenen Internetpräsenz. Diese Seite wurde mittels GETESS–Gathering–Prozeß analysiert. Das resultierende abstract kann im Anhang D eingesehen werden.
Die zweite HTML–Seite ist eine Aufzählung von Hotels in und um Bad Doberan. Die Schwierigkeit

[2] Der Internetzugriff auf die einzusammelnde Seite wird weiterhin benötigt.

hier besteht in der Erkennung, daß es sich um eine Übersichtsseite mit mehreren Instanzen des Hotel–Konzeptes handelt. Dabei ist die Klassenerkennung entscheidend. Das Ergebnis ist ebenfalls im Anhang D dargestellt.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurde gezeigt, wie durch die Kombination verschiedener Analysekonzepte aus verschiedenen Forschungsgebieten sinnvolle Informationsextraktions–Werkzeuge aufgebaut werden können.

Inhaltliche Zusammenfassung Ausgehend von herkömmlichen Suchmaschinen wurde das Projekt GETESS als eine mögliche und sinnvolle Verbesserung der bisherigen Internet–Suche vorgestellt und in bestehende Internet–Agentensysteme eingeordnet.
Es wurden die extrahierbaren Internet–Daten identifiziert und klassifiziert. Eine mögliche Auswahl der Daten wurde nach bestimmten Gesichtspunkten vorgenommen und Fragen der Modellierung diskutiert. Diese Daten bilden die Grundlage für die hier vorgestellte Informationsextraktion.
Als interessanten Aspekt stellten sich domänenspezifisch und linguistisch motivierte Daten dar. Eine Toolbox, bestehend aus einer linguistischen Komponente und einer domänenspezifischen Komponente, wurde explizit vorgestellt.
Die Kombination der Techniken zur Gewinnung der Daten und die sinnvolle Verarbeitung haben sich als Hauptproblem dieser Arbeit herausgestellt. In Kapitel 4 wurde eine konzeptionelle Idee der Umsetzung des Informationsextraktionsproblems dargelegt.
Die prototypische Umsetzung des Konzeptes ermöglicht nun eine Verbesserung der Informationsextraktion im GETESS–Projekt. Am Beispiel wird diese Umsetzung in Kapitel 5 gezeigt.
Eine genaue Bewertung hat gezeigt, daß die erarbeitete Konzeption prinzipiell funktioniert und für zukünftige WWW–Applikationen interessant sein kann.

Erreichte Ziele Folgende konkrete Ziele konnten mit dieser Arbeit erreicht werden:

- Konzeption und Realisierung einer Informationsextraktion auf Grundlage von strukturellen, linguistischen und domänenspezifischen Analysetechniken
- Extraktion und Kombination semantischer bzw. inhaltlicher Daten von Dokumenten und Dokumentgruppen
- Klassifizierung der Dokumente mit Hilfe der Techniken aus der konzipierten Informationsextraktion
- Konzeption der Anwendung der beschriebenen Analysetechniken zur Erkennung irrelevanter Dokumentdaten bezüglich der Anwendungsdomäne
- Vorschlag einer konkreten Realisierung im GETESS–Gathering

6.2 Ausblick

Es werden nun abschließend einige interessante Aspekte gezeigt, die die hier vorgestellten Konzepte ergänzen und neue Perspektiven ermöglichen.

Eine interessante Sichtweise ergibt sich aus der Generalisierung des Problems der Informationsextraktion von semistrukturierten Textdaten. Um den Inhalt solcher Textdaten zu erschließen, benötigt man Techniken zur Analyse der Dokumentstruktur und zur Analyse der Texte. Die Idee ist, daß eine kombinierte Analyse mit zum Teil unterschiedlich gewichteten Analyseschritten eine verbesserte Inhaltsextraktion ermöglichen kann. Eine formale Beschreibung solcher Gewichtungen und der resultierenden Kombination wäre dazu sinnvoll.

Spezialisiert man die hier vorgestellten Konzepte auf ein bestimmtes Anwendungsszenario ergeben sich ganz spezielle Analyse- und Verarbeitungswerkzeuge. Solche abgestimmten Anwendungsszenarios enthalten sinnvolle Einschränkungen, die die Komplexität der Inhaltserschließung vereinfachen können. Das bedeutet, daß die in einer Anwendung auftretenden Spezialfälle interpretiert und die allgemeinen Tools spezialisiert und verbessert werden können.
Spezialisierungen könnten sein:

- spezielle Dokumenttypen → spezielle Strukturanalyse, angepaßte, linguistische Analyse
- spezielle Anwendungsdomäne → spezielle Ontologie
- spezielles Anwendungsszenario → Reihenfolge und Gewichtung der Analyseschritte
- spezielle Zielstruktur → spezielle Dokument–abstract–Abbildung

Einige Aspekte sind in dieser Arbeit offen geblieben.
Die Erkennung von redundanten Informationen ist nicht allein auf den Gathering–Prozeß beschränkt. Innerhalb kleinerer Sammelumgebungen könnte die Informationsextraktion diese Aufgabe übernehmen. Bei großen, eventuell verteilten Datenbeständen hat nur noch die Datenbasis mit speziellen Datenbanksystemen die Möglichkeit globale Redundanzen zu finden. Die Techniken für ein solches umfassendes Redundanzfinden kann möglicherweise eine Kombination aus Datenbanktechniken und den hier vorgestellten Gathering–Techniken darstellen.
Die Möglichkeiten irrelevante Daten aus dem Dokument zu entfernen, bevor die abstracts gebildet werden, könnten verbessert werden, indem man versucht, durch Dokumentvergleiche innerhalb der Dokumente eines Servers Strukturen wie die Copyright–Angaben oder wiederkehrende, allgemeine Seitennavigationsstrukturen zu erkennen und zu entfernen.
Ein sehr wichtiger, offen gebliebener Aspekt ist das Update der Seiten. Im Idealbeispiel in Kapitel 4.4 werden einige Daten, wie Update–Zeiten und Prüfsummen, für diesen Zweck vorgehalten. Weiterhin unterstützt Harvest ein inkrementelles Update. Das Problem ist allerdings, daß bei einem Update eines veränderten Dokuments die abhängigen abstracts möglicherweise neu gebildet werden müssen. Da allerdings einige abstracts durch mehrere Dokumente gebildet werden können, kann sich bei Änderung eines Dokuments eine ganze Welle von Dokumenten–Updates durch die Datenbasis entwickeln. Die einfachste Lösung wäre, einfach ein komplettes Update der Datenbasis durchzuführen. Es ist allerdings sinnvoll, bei sehr großen Datenbeständen ein partielles oder inkrementelles Update durch den GETESS–Gatherer zu entwickeln.

In der Arbeit wurde in erster Linie von der bestehenden HTML–WWW–Struktur ausgegangen. Es ist sinnvoll, einen Ausblick in die Zukunft des Hypertext–Internet zu machen.
Zum einen ist interessant, daß mehr und mehr Daten dynamisch im Web vorzufinden sind. Dies betrifft vor allem große Datenbestände, die meist in Datenbanken gehalten und mittels Datenbanksprachen angefragt werden. Bei unbekanntem Schema und unbekanntem Inhalt ist es sehr schwierig die relevanten Informationen zu erschließen. Eine Möglichkeit könnte eventuell durch die in dieser Arbeit vorgestellten Konzepte entstehen, indem der Inhalt der Zugriffsseite und deren Internet–Umgebung ermittelt und durch domänenspezifisches Wissen mögliche Anfragen generiert werden.

Zum anderen könnte sich ein Großteil des Internet nach und nach auf das neue, vom W3C (World Wide Web Consortium) etablierte XML–Format umstellen. Wenn die vorgeschlagenen XML–Konzepte strikt eingehalten werden sollten, würden sich einige Vereinfachungen bei der Struktur- und Kontextanalyse (siehe Kapitel 5) ergeben.
Weiterhin ist anzumerken, daß bei dem derzeitigen Wachstum der Internetinformationen eine Spezialisierung der Suchmaschinen anzuraten ist. Dabei darf man nicht aus den Augen verlieren, daß die Daten mehrfach vorhanden, falsch, kurzlebig oder langlebig vorliegen können. Solche Informationen werden vorläufig nur mittels manuellem bzw. semiautomatischen Eingriff mit einer hohen Wahrscheinlichkeit erkennbar sein.

Im Projekt GETESS ist geplant, eine zweite Ontologie aufzubauen. Bei dieser zweiten Ontologie werden die gewonnen Knowledge–Tools zur Umstellung auf die neue Ontologie eingesetzt. Für die hier vorgestellten Konzepte ergeben sich daraus mehr oder weniger Umstellungen auf ein neues Anwendungsszenario (siehe oben).
Wenn allerdings ein Anwendungsszenario mit zwei oder mehr Ontologien gleichzeitig gewünscht ist, wird in der hier vorgestellten Konzeption mindestens ein weiterer Zwischenschritt nötig sein. Bevor die linguistisch, domänenspezifische Analyse gestartet wird, sollte die Ontologiedomäne bestimmt werden. Dann wird es für jede Domäne eine andere Datenanalyse bzw. Datenverarbeitung geben. Ist aufgrund von Mehrdeutigkeiten die Ontologiedomäne einer Seite nicht bestimmbar, könnte nach allen festgestellten Domänen analysiert werden. Das Ergebnis wird wahrscheinlich weniger qualitativ hochwertig sein als bei einem für eine Domäne optimiertem System. Trotzdem ist dieses Anwendungungsszenario im Internet nicht auszuschließen.

Literaturverzeichnis

[AB00] Aas, Gisle und Sean M. Burke: *Perl-Bibliothek: HTML-Tree-0.66.* http://www.cpan.org/HTML-Tree-0.66.tar.gz, 2000.

[AC00] Aas, Gisle und Michael A. Chase: *Perl-Bibliothek: HTML-Parser-3.08.* http://www.cpan.org/HTML-Parser-3.08.tar.gz, 2000.

[Ade98] Adelberg, Brad: *NoDoSE – A Tool for Semi–Automatically Extracting Structured and Semistructured Data from Text Documents.* In: Haas, L. und A. Tiwary (Herausgeber): *SIGMOD'98, Proc. of the 1998 ACM SIGMOD Int. Conf. on Management of Data*, Nummer 25 in *ACM SIGMOD Record*, Seiten 283–294, Seattle, USA, June 1998. ACM Press.

[AQM+96] Abiteboul, Serge, Dallan Quass, Jason McHugh, Jennifer Widom und Janet L. Wiener: *The Lorel Query Language for Semistructured Data.* Technical Report, Department of Computer Science, Stanford University, 1996.

[BDB+00] Bruder, Ilvio, Antje Düsterhöft, Markus Becker, Jochen Bedersdorfer und Günter Neumann: *GETESS: Constructing a Linguistic Search Index for an Internet Search Engine.* Technical Report, University of Rostock and DFKI GmbH Saarbrücken, March 2000.

[BM98] Behme, Henning und Stefan Mintert: *Extensible Markup Language (XML) — in der Praxis.* Addison Wesley Longman Verlag GmbH, Bonn, 1998.

[BPW+98] Bruder, Ilvio, Jan Pretzel, Burkhard Wrenger, Bernd Prager, Günter Neumann, Christian Braun, Hans Uszkoreit, Hans-Peter Schnurr, Steffen Staab, Rudi Studer, Antje Düsterhöft, Meike Klettke und Andreas Heuer: *BMBF–Projekt GETESS: GErman Text Exploration and Search System — Arbeitsbericht zum Meilenstein 1.* Meilensteinbericht 1, GECKO mbH Rostock, DFKI Saarbrücken, Universität Karlsruhe und Universität Rostock, Dezember 1998.

[Bru99] Bruder, Ilvio: *Datenbanktechniken für das WWW.* Hauptseminar, Fachbereich Informatik, Univ. Rostock, April 1999.

[Bru00] Bruder, Ilvio: *Integration von SMES und Harvest in GETESS.* Studienarbeit, Universität Rostock, Fachbereich Informatik, Januar 2000.

[Bus90] Bussmann, Hadumod: *Lexikon der Sprachwissenschaft.* Alfred Kröner Verlag, zweite, neubearb. Auflage, 1990.

[BYRN99] Baeza-Yates, Ricardo und Berthier Ribeiro-Neto: *Modern Information Retrieval.* ACM Press. Addison Wesley, New York, 1999.

[BZW98] Brenner, Walter, Rüdiger Zarnekow und Hartmut Wittig: *Intelligente Softwareagenten.* Springer Verlag, Berlin, 1998.

[CGMH+94] Chawathe, Sudarshan, Hektor Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullmann und Jennifer Widom: *The TSIMMIS Project: Integration of Heterogenous Information Sources.* In: *Proc. of IPSI Conf.*, 1994.

[DG00] Düsterhöft, Antje und Sherry Gröticke: *A Heuristic Approach for Recognizing a Document's Language Used for the Internet Search Engine GETESS.* In: *Proc. 2nd Int. Workshop on NLIS 2000*, London, 2000.

[FDES97] Fensel, Dieter, Stefan Decker, Michael Erdmann und Rudi Studer: *Ontobroker: Transforming the WWW into a Knowledge Base.* Technical Report, University of Karlsruhe, Institute AIFB, Karlsruhe, Oktober 1997.

[FFLS97] Fernandez, Mary, Daniela Florescu, Alon Levy und Dan Suciu: *A Query Language and Processor for a Web-Site Management System.* Technical Report, Inria Roquencourt and Univ. of Washington, April 1997.

[FLM98] Florescu, Daniela, Alon Levy und Alberto Mendelzon: *Database Techniques for the World–Wide Web: A Survey.* Technical Report, Inria Roquencourt, Univ. of Washington and Univ. of Toronto, 1998.

[Fuh97] Fuhr, Norbert: *Information Retrieval, Skriptum zur Vorlesung.* Vorlesungsskript, Universität Dortmund, Dortmund, Mai 1997. Kap. 1-3 Einführung und Konzepte, Kap. 9 IR and Databases.

[GW93] Goerz, G. und W. Wahlster: *Sprachverarbeitung: Einleitung und Überblick.* In: Goerz, G. (Herausgeber): *Einführung in die künstliche Intelligenz.* Addison Wesley, Bonn, 1993.

[HMW99] Heuer, Andreas, Holger Meyer und Gunnar Weber: *SWING: Die Suchmaschine des Landesinformationssystems MV-Info.* Technischer Bericht, Universität Rostock, Fachbereich Informatik, Rostock, Juni 1999.

[HP99] Heuer, Andreas und Denny Priebe: *IRQL –Yet Another Language for Quering Semi–Structured Data.* Preprint CS-01-99, Universität Rostock, Fachbereich Informatik, 1999.

[HP00] Heuer, Andreas und Denny Priebe: *Integrating a Query Language for Structured and Semi-Structured Data and IR Techniques.* In: *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA 2000).* IEEE Computer Society Press, September 2000.

[HSW96] Hardy, Darren R., Michael F. Schwartz und Duane Wessels: *Harvest Users Manual.* University of Colorado at Boulder, Department of Computer Science, January 1996.

[LDHM97] Langer, Uwe, Antje Düsterhöft, Andreas Heuer und Holger Meyer: *SWING: Ein Anfrage- und Suchdienst im Internet.* Rostocker Informatik–Bericht, Fachbereich Informatik, Universität Rostock, Rostock, 1997.

[Los98] Losee, Robert M.: *Text Retrieval and Filtering: Analytical Models of Performance.* Kluwer Academic Publishers, Boston, 1998.

[MAG+97] McHugh, Jason, Serge Abiteboul, Roy Goldman, Dallan Quass und Jennifer Widom: *Lore: A Database Management System for Semistructured Data.* ACM SIGMOD Record 26 (3), Seiten 54–72, 1997.

[MAM+98] Mecca, G., P. Atzeni, P. Merialdo, A. Masci und G. Sindoni: *Frome Databases to Web-Bases: The ARANEUS Experience.* Technical Report RT-DIA-34-1998, Dipartimento di Informatica e Automazione, Universita di Roma Tre, May 1998.

[MPP+99] Melzig, Siegfried, Bernd Prager, Jan Pretzel, Burkhard Wrenger, Günter Neumann, Christian Braun, Hans Uszkoreit, Alexander Mädche, Hans-Peter Schnurr, Steffen Staab, Rudi Studer, Antje Düsterhöft, Meike Klettke, Denny Priebe und Andreas Heuer: *BMBF–Projekt GETESS: GErman Text Exploration and Search System — Arbeitsbericht zum Meilenstein 2.* Meilensteinbericht 2, GECKO mbH Rostock, DFKI Saarbrücken, Universität Karlsruhe und Universität Rostock, Juli 1999.

[NBB+97] Neumann, Günter, Rolf Backofen, Judith Baur, Markus Becker und Christian Braun: *An Information Extraction Core System for Real World German Text Processing.* In: *Proc. of the ANLP'97*, Seiten 208–215, Washington, USA, March 1997.

[NM98] Neumann, Günter und Giampaolo Mazzini: *Domain–adaptive Information Extraction.* Draft Version, DFKI Saarbrücken, Saarbrücken, Germany, August 1998.

[PGMW95] Papakonstantinou, Yannis, Hector Garcia-Molina und Jennifer Widom: *Object Exchange Across Heterogeneous Information Sources.* Technical Report, Stanford University, Department of Computer Science, 1995.

[SA99] Sahuguet, A. und F. Azavant: *Building Light-Wight Wrappers for Legacy Web Data-Sources Using W4F.* In: Atkinson, M. P., M. E. Orlowska, P. Valduriez, S. B. Zdonik und M. L. Brodie (Herausgeber): *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*, Seiten 738–741, Edinburgh, UK, September 1999. Morgan Kaufmann.

[sea00a] *Search Engines Shoot-out - Top Engines Compared.* http://coverage.cnet.com/Content/Reviews/Compare/Search2/, 2000.

[sea00b] *Understanding and Comparing Web Search Tools.* http://web.hamline.edu/administration/libraries/search/comparisons.html, 2000.

[sew00] *Search Engine Watch.* http://www.searchenginewatch.com, 2000.

[Tol97] Tolksdorf, Robert: *Die Sprache des Web: HTML 4.* dpunkt Verlag, Heidelberg, dritte Auflage, 1997.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

HTML–Beispielseite

Die hier beschriebene HTML–Seite ist stellvertretend für ein momentan allgemein so dargestelltes Dokument in der Tourismusbranche in Mecklenburg/Vorpommern. Die Seite wurde dem Server "www.all-in-all.de", einem Tourismusdaten–Provider in Mecklenburg/Vorpommern, entnommen.
In diesem Kapitel werden das Aussehen dieser Seite in einem Browser, der HTML–Code und die Baumstruktur der Seite dargelegt.
Zunächst zeigt die Abbildung A.2 die Browser–Darstellung der Seite. Es handelt sich dabei um eine Hotelseite mit Lageinformationen, Zimmerbeschreibung, Preisen und Ausstattungsmerkmalen.

HTML–Seiten sind in einen Header und einen Body unterteilt. Dazu gibt es diverse Tags (eingeklammerte Befehlsworte), mit denen man die Ordnung und das Layout der Seite bestimmt.
Im folgenden ist die HTML–Seite mit einigen erklärenden Zwischenworten beschrieben. Für eine allgemeine Erklärung des HTML–Codes sei auf [Tol97] verwiesen.

```
<html>
```

Beginn Header.

```
<head>
```

Einige Metadaten.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="author" content="MANET Marketing GmbH, Mecklenburg- Vorpommern und die
Ostsee">
<meta name="description" content="Das Atrium Hotel Krüger in Rostock - Mecklenburg
erleben - Alles in einem! - Informationen sinnvoll vernetzt.MANET">
<meta name="expires" content="07 Sep. 2000 12:00:00 GMT">
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
<meta name="keywords" content="Rostock, Mecklenburg, Tourismus, Tagung, Hotels,
Zimmer, Reservierung, Hansestadt, Ostsee, Küste, Atrium Hotel Krüger">
<meta name="lang" content="de, ch, at">
<meta name="revisit-after" content="31 days">
<meta name="robots" content="all">
<meta name="timestamp" content="990907 163644">
```

Der Titel des Dokuments.

```
<title>Das Atrium Hotel Krüger in Rostock - Urlaub und Reisen in Mecklenburg
</title>
```

Ein JavaScript–Programm mit Unterscheidung zwischen verschiedenen Browsern.

```
<script language="JavaScript"><!--
<!--
function PopUp()
{

if (navigator.appName == 'Netscape')
{
var url='bilderbogen/b1127.htm';
var win='left=290,top=150,toolbar=0,directories=0,menubar=0,scrollbars=0,
resizable=0,width=369,height=249';
open(url,'banner', win);
}
else //internet explorer oder andere
{var url='bilderbogen/b1127.htm';
var win='left=290,top=150,toolbar=0,directories=0,menubar=0,scrollbars=0,
resizable=0,width=350,height=234';
open(url,'banner', win);
}
}
// --></script>
```

Ende des Headers.

```
</head>
```

Beginn des Bodies mit einigen Grundeinstellungen zur Farbe etc.

```
<body background="grund.gif" bgcolor="#F7F7ED" text="#000000" link="#008000"
vlink="#008000" alink="#FF0000">
```

Dieser Block ist als Tabelle dargestellt und wird zur Darstellung von Icons in einer bestimmten Anordnung benutzt.

```
<div align="center"><center>
<table border="0" cellpadding="2" width="620">
<tr>
<td><p align="left"><a href="index.htm" target="_top"><img src="start.gif"
alt="Tourismus-Information - Startseite" align="left" border="0" hspace="0"
WIDTH="148" HEIGHT="24"></a></td>
<td><p align="right">[<a href="suche.htm">Suche</a>] [<a href="english/1127.htm">
English</a>]</td></tr>
</table>
</center></div>
```

Dieser Block bezeichnet dieselbe Funktion wie der vorherige.

```
<div align="center"><center>
<table border="0" cellpadding="2" width="640">
<tr>
<td><font size="4"><b><a href="http://www.all-in-all.com/fax/fax.cgi?form=fax2&amp
;firmid=F2ATRIUMROSTOCK"><img src="fax.gif" alt="Kontaktaufnahme, Buchungsanfrage,
Atrium Hotel Krüger" border="0" WIDTH="90" HEIGHT="30"></a><a href="tb030.htm">
<img src="esstrink.gif" alt="Essen &amp; Trinken, Hansestadt Rostock, Mecklenburg"
border="0" WIDTH="90" HEIGHT="30"></a><a href="9122.htm"><img src="wassersp.gif"
alt="Wassersport, Hansestadt Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT=
"30"></a><a href="tb042.htm"><img src="sportfit.gif" alt="Sport &amp; Fitneß,
Hansestadt Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT="30"></a><a href=
"tb048.htm"><img src="kulturkl.gif" alt="Kultur, Hansestadt Rostock, Mecklenburg"
border="0" WIDTH="90" HEIGHT="30"></a><a href="9192.htm"><img src="sehenskl.gif"
alt="Sehenswertes, Hansestadt Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT=
"30"></a><a href="9138.htm"><img src="touren.gif" alt="Angebote, Hansestadt
Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT="30"></a></b></font></td>
</tr>
</table>
</center></div>
```

Dieser Block ist wieder als Tabelle dargestellt. Hier findet man teilweise Bilder und die Hauptinformationen der Seite, mittig dargestellt.

```
<div align="center"><center>
<table border="0" cellpadding="2" cellspacing="7" width="620">
<tr>
<td align="center"><font size="4" face="Arial"><img src="bild/b1127_logo.gif" alt=
"Logo Atrium Hotel Krüger" align="left" hspace="10" vspace="5" WIDTH="120" HEIGHT=
"80"></font><font size="4"><br>
</font></td>
<td valign="top"><p align="center"><b><i><font size="5" color="#000080">Atrium
Hotel Krüger </font><font size="1" color="#000080"><br>
</font></i></b><font size="1"><br>
</font><font color="#000040" size="3">D-18069 Rostock-Sievershagen <br>
Ostsee-Park-Str. 2 <br>
Tel:(0381)800 23 43  Fax:(0381)800 23 42 </font></td>
<td align="center"><b><a href="javascript:PopUp()"><font size="4" face="Arial">
<img src="bild/b1127_1k.gif" alt="Atrium Hotel Krüger" align="left" border="0"
hspace="10" vspace="5" WIDTH="120" HEIGHT="80"></font></a><a href="g43g.htm"><font
size="4"><br>
</font></a></b></td></tr>
</table>
</center></div>
```

Ein weiterer Block, als Tabelle dargestellt, enthält die eigentlichen Informationen dieses Dokumentes.

```
<div align="center"><center>
<table border="0" cellpadding="2" cellspacing="7" width="620">
<tr><td><hr></td></tr>
<tr>
<td valign="top"><b><font size="3" color="#000080">Lage: </font><font color=
```

```
"#000040" size="3"><br>
</b>Das von der Familie Krüger geführte GARNI-HOTEL im Westen der</font><font
size="3"> </font><a href="1022.htm"><font color="#008000" size="3"><strong>
Hansestadt Rostock</strong></font></a><font size="3"> </font><font color="#000040"
size="3">präsentiert sich als ein modernes Geschäfts- und Messehotel. <br>
Touristen oder Geschäftsreisende schätzen die angenehme und freundliche Atmosphäre.
</font><b><font size="3"><br>
<font color="#000080">Hausbeschreibung: </font></font><font color="#000040" size=
"3"><br>
</b>Die 59 Komfortzimmer und Appartements sind mit Bad oder Dusche/WC, Telefon,
TV, <b>Radio, </b>Minibar und Fön ausgestattet. </font><font size="3"><br>
<b><font color="#000080">Preise:</font> </font><font color="#000040" size="3">
(inclusive Frühstück) <br>
</b>Einzelzimmer: von 95,00 - 125,00 DM  Doppelzimmer: 150,00 DM, <br>
Dreibettzimmer: 185,00 DM  Appartements: von 135,00 - 190,00 DM </font></td>
</tr>
<tr>
<td valign="top"><font color="#000080" size="3"><b>Bei uns willkommen: </b></font>
<font color="#000040" size="3">American-Express, Visa-Card, Euro-Card, Diners-Club
</font><font size="3"> </font></td>
</tr>
<tr>
<td valign="top"><font size="3"><b><font color="#000080">Besonderheiten:</font>
<br></b></font>
<font color="#000040" size="3">Tagungen und Seminare können bis zu einer Kapazität
von 40 Personen durchgeführt werden. Das Haus ist idealer Ausgangspunkt, um den
Ostseestrand (5 km) und die reizvolle Umgebung bequem zu erreichen. </font></td>
</tr>
</table>
</center></div>
```

Zum Schluß sind hier noch Informationen zum Provider und zum Copyright.

```
<p align="center"><img src="copy.gif" WIDTH="555" HEIGHT="20"><br>
<i><font size="2">Copyright (c) <!--webbot bot="Timestamp" s-type="REGENERATED"
s-format="%B %Y" startspan -->September 1999<!--webbot bot="Timestamp" endspan
i-checksum="29531" --> </font><a href="0702.htm">MANET</a><font size="2">
Marketing GmbH, Schwerin; Fax: +49-(0)385-3993-411</i> </font></p>
```

Ende des Bodies und des Dokumentes.

```
</body>
</html>
```

Um die Struktur dieser HTML–Seite besser darzustellen, wird nun die HTML–Baumstruktur (Abbildung A.1) gezeigt. Zwecks Übersichtlichkeit wird nur ein zusammengefaßter Baum ab dem body–Tag gezeigt.

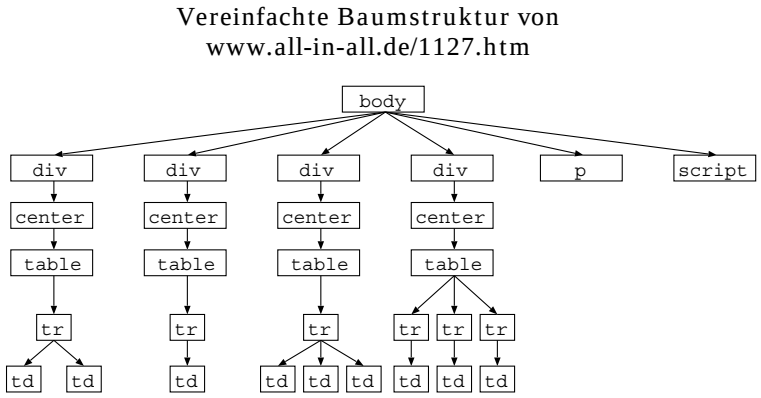

Abbildung A.1: HTML–Strukturbaum der Seite “1127.htm” aus “www.all-in-all.de”

Abbildung A.2: HTML–Seite “1127.htm” vom Server “www.all-in-all.de”

Anhang B

Bisheriges Ergebnis–abstract der HTML–Beispielseite

Die HTML–Beispielseite von Anhang A "www.all-in-all.de/1127.htm" wurde mit der bisherigen Analyse bearbeitet. Das folgende Ergebnis–abstract ist demnach ohne Strukturanalyse und Separierung von Informationspaketen entstanden.
Es zeigt vor allem Probleme bei der Relationierung der domänenspezifischen Konzepte. Es wird dabei jedes Konzept mit jedem innerhalb eines Satzes relationiert, und es werden alle Konzepte des Titels mit allen weiteren Konzepten im gesamten Dokument in Beziehung gesetzt. Eine genauere Darstellung der Probleme findet sich in 3.3.

```
<list>
<tuple name='getess-output'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
```

```
</tuple><tuple name='getess-output'><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
```

```
    <name>telefon</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Person</type><inst>Person_1</inst>
    <name>person</name>
  </fragment><fragment>
    <type>Person</type><inst>Person_1</inst>
    <name>40</name>
  </fragment>
  <constraint type='HEURISTIC'>(CARD)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_2</inst>
    <name>hotel</name>
  </fragment><fragment>
```

```
    <type>Hotel</type><inst>Hotel_2</inst>
    <name>krueger</name>
  </fragment>
  <constraint type='HEURISTIC'>(MODIFIER)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>krueger</name>
  </fragment>
  <constraint type='HEURISTIC'>(MODIFIER)</constraint>
</tuple><tuple name='getess-output' rel='liegt_in_Stadt'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC
                                MY-NP-PP-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Stadt</type><inst>schwerin</inst>
    <name>Schwerin</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Unterkunft</type><inst>Unterkunft_1</inst>
    <name>haus</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Person</type><inst>Person_1</inst>
    <name>person</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type>
    <inst>mecklenburg</inst><name>Mecklenburg</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
```

```
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Tourist</type><inst>Tourist_1</inst>
    <name>tourist</name>
  </fragment>
```

```
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_2</inst>
    <name>hotel</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Stadt</type><inst>schwerin</inst>
    <name>Schwerin</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Unterkunft</type><inst>Unterkunft_1</inst>
    <name>haus</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Person</type><inst>Person_1</inst>
    <name>person</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
```

```
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Tourist</type><inst>Tourist_1</inst>
    <name>tourist</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
```

```
  </fragment><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_2</inst>
    <name>hotel</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output' rel='liegt_in_Stadt'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Unterkunft</type><inst>Unterkunft_1</inst>
    <name>haus</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Person</type><inst>Person_1</inst>
    <name>person</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
```

```
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Tourist</type><inst>Tourist_1</inst>
    <name>tourist</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output' rel='liegt_in_Stadt'><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_2</inst>
    <name>hotel</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output' rel='liegt_in_Stadt'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Stadt</type><inst>schwerin</inst>
    <name>Schwerin</name>
  </fragment>
```

```
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Unterkunft</type><inst>Unterkunft_1</inst>
    <name>haus</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Person</type><inst>Person_1</inst>
    <name>person</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
```

```
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Tourist</type><inst>Tourist_1</inst>
    <name>tourist</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple><tuple name='getess-output' rel='liegt_in_Stadt'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment>
  <constraint type='HEURISTIC'>(TITLE-HEURISTIC)</constraint>
</tuple>
</list>
```

Anhang C

Analyseergebnisse der Beispielseite

In diesem Abschnitt werden die einzelnen Zwischenergebnisse der Analyse und Verarbeitung der HTML–Seite aus Anhang A dargestellt. Erklärungen zu den Analyseschritten finden sich in Kapitel 5.
Zunächst das Ergebnis der HTML–Strukturanalyse ohne Meta–Tags und Programmierschnittstellen–Tags:

```
<ftree>
<node pos='0' tagname='html'/>
<node pos='0.0' tagname='head'/>
<node pos='0.0.0' tagname='title'/>
<node pos='0.1' tagname='body'/>
<node pos='0.1.0' tagname='div'/>
<node pos='0.1.0.0' tagname='center'/>
<node pos='0.1.0.0.0' tagname='table'/>
<node pos='0.1.0.0.0.0' tagname='tr'/>
<node pos='0.1.0.0.0.0.0' tagname='td'/>
<node pos='0.1.0.0.0.0.0.0' tagname='p'/>
<node pos='0.1.0.0.0.0.0.0.0' tagname='a'/>
<node pos='0.1.0.0.0.0.0.0.0.0' tagname='img'/>
<node pos='0.1.0.0.0.0.1' tagname='td'/>
<node pos='0.1.0.0.0.0.1.0' tagname='p'/>
<node pos='0.1.0.0.0.0.1.0.1' tagname='a'/>
<node pos='0.1.0.0.0.0.1.0.3' tagname='a'/>
<node pos='0.1.1' tagname='div'/>
<node pos='0.1.1.0' tagname='center'/>
<node pos='0.1.1.0.0' tagname='table'/>
<node pos='0.1.1.0.0.0' tagname='tr'/>
<node pos='0.1.1.0.0.0.0' tagname='td'/>
<node pos='0.1.1.0.0.0.0.0' tagname='font'/>
<node pos='0.1.1.0.0.0.0.0.0' tagname='b'/>
<node pos='0.1.1.0.0.0.0.0.0.0' tagname='a'/>
<node pos='0.1.1.0.0.0.0.0.0.0.0' tagname='img'/>
<node pos='0.1.1.0.0.0.0.0.0.1' tagname='a'/>
<node pos='0.1.1.0.0.0.0.0.0.1.0' tagname='img'/>
<node pos='0.1.1.0.0.0.0.0.0.2' tagname='a'/>
<node pos='0.1.1.0.0.0.0.0.0.2.0' tagname='img'/>
<node pos='0.1.1.0.0.0.0.0.0.3' tagname='a'/>
<node pos='0.1.1.0.0.0.0.0.0.3.0' tagname='img'/>
<node pos='0.1.1.0.0.0.0.0.0.4' tagname='a'/>
<node pos='0.1.1.0.0.0.0.0.0.4.0' tagname='img'/>
<node pos='0.1.1.0.0.0.0.0.0.5' tagname='a'/>
```

```
<node pos='0.1.1.0.0.0.0.0.5.0' tagname='img'/>
<node pos='0.1.1.0.0.0.0.0.6' tagname='a'/>
<node pos='0.1.1.0.0.0.0.0.6.0' tagname='img'/>
<node pos='0.1.2' tagname='div'/>
<node pos='0.1.2.0' tagname='center'/>
<node pos='0.1.2.0.0' tagname='table'/>
<node pos='0.1.2.0.0.0' tagname='tr'/>
<node pos='0.1.2.0.0.0.0' tagname='td'/>
<node pos='0.1.2.0.0.0.0.0' tagname='font'/>
<node pos='0.1.2.0.0.0.0.0.0' tagname='img'/>
<node pos='0.1.2.0.0.0.0.1' tagname='font'/>
<node pos='0.1.2.0.0.0.0.1.0' tagname='br'/>
<node pos='0.1.2.0.0.0.1' tagname='td'/>
<node pos='0.1.2.0.0.0.1.0' tagname='p'/>
<node pos='0.1.2.0.0.0.1.0.0' tagname='b'/>
<node pos='0.1.2.0.0.0.1.0.0.0' tagname='i'/>
<node pos='0.1.2.0.0.0.1.0.0.0.0' tagname='font'/>
<node pos='0.1.2.0.0.0.1.0.0.0.1' tagname='font'/>
<node pos='0.1.2.0.0.0.1.0.0.0.1.0' tagname='br'/>
<node pos='0.1.2.0.0.0.1.0.1' tagname='font'/>
<node pos='0.1.2.0.0.0.1.0.1.0' tagname='br'/>
<node pos='0.1.2.0.0.0.1.0.2' tagname='font'/>
<node pos='0.1.2.0.0.0.1.0.2.1' tagname='br'/>
<node pos='0.1.2.0.0.0.1.0.2.3' tagname='br'/>
<node pos='0.1.2.0.0.0.2' tagname='td'/>
<node pos='0.1.2.0.0.0.2.0' tagname='b'/>
<node pos='0.1.2.0.0.0.2.0.0' tagname='a'/>
<node pos='0.1.2.0.0.0.2.0.0.0' tagname='font'/>
<node pos='0.1.2.0.0.0.2.0.0.0.0' tagname='img'/>
<node pos='0.1.2.0.0.0.2.0.1' tagname='a'/>
<node pos='0.1.2.0.0.0.2.0.1.0' tagname='font'/>
<node pos='0.1.2.0.0.0.2.0.1.0.0' tagname='br'/>
<node pos='0.1.3' tagname='div'/>
<node pos='0.1.3.0' tagname='center'/>
<node pos='0.1.3.0.0' tagname='table'/>
<node pos='0.1.3.0.0.0' tagname='tr'/>
<node pos='0.1.3.0.0.0.0' tagname='td'/>
<node pos='0.1.3.0.0.0.0.0' tagname='hr'/>
<node pos='0.1.3.0.0.1' tagname='tr'/>
<node pos='0.1.3.0.0.1.0' tagname='td'/>
<node pos='0.1.3.0.0.1.0.0' tagname='b'/>
<node pos='0.1.3.0.0.1.0.0.0' tagname='font'/>
<node pos='0.1.3.0.0.1.0.0.1' tagname='font'/>
<node pos='0.1.3.0.0.1.0.0.1.0' tagname='br'/>
<node pos='0.1.3.0.0.1.0.2' tagname='font'/>
<node pos='0.1.3.0.0.1.0.3' tagname='a'/>
<node pos='0.1.3.0.0.1.0.3.0' tagname='font'/>
<node pos='0.1.3.0.0.1.0.3.0.0' tagname='strong'/>
<node pos='0.1.3.0.0.1.0.4' tagname='font'/>
<node pos='0.1.3.0.0.1.0.5' tagname='font'/>
<node pos='0.1.3.0.0.1.0.5.1' tagname='br'/>
<node pos='0.1.3.0.0.1.0.6' tagname='b'/>
<node pos='0.1.3.0.0.1.0.6.0' tagname='font'/>
<node pos='0.1.3.0.0.1.0.6.0.0' tagname='br'/>
<node pos='0.1.3.0.0.1.0.6.0.1' tagname='font'/>
<node pos='0.1.3.0.0.1.0.6.1' tagname='font'/>
<node pos='0.1.3.0.0.1.0.6.1.0' tagname='br'/>
<node pos='0.1.3.0.0.1.0.8' tagname='b'/>
```

```
<node pos='0.1.3.0.0.1.0.10' tagname='font'/>
<node pos='0.1.3.0.0.1.0.10.0' tagname='br'/>
<node pos='0.1.3.0.0.1.0.10.1' tagname='b'/>
<node pos='0.1.3.0.0.1.0.10.1.0' tagname='font'/>
<node pos='0.1.3.0.0.1.0.11' tagname='font'/>
<node pos='0.1.3.0.0.1.0.11.1' tagname='br'/>
<node pos='0.1.3.0.0.1.0.11.3' tagname='br'/>
<node pos='0.1.3.0.0.2' tagname='tr'/>
<node pos='0.1.3.0.0.2.0' tagname='td'/>
<node pos='0.1.3.0.0.2.0.0' tagname='font'/>
<node pos='0.1.3.0.0.2.0.0.0' tagname='b'/>
<node pos='0.1.3.0.0.2.0.1' tagname='font'/>
<node pos='0.1.3.0.0.2.0.2' tagname='font'/>
<node pos='0.1.3.0.0.3' tagname='tr'/>
<node pos='0.1.3.0.0.3.0' tagname='td'/>
<node pos='0.1.3.0.0.3.0.0' tagname='font'/>
<node pos='0.1.3.0.0.3.0.0.0' tagname='b'/>
<node pos='0.1.3.0.0.3.0.0.0.0' tagname='font'/>
<node pos='0.1.3.0.0.3.0.0.0.1' tagname='br'/>
<node pos='0.1.3.0.0.3.0.1' tagname='font'/>
<node pos='0.1.4' tagname='p'/>
<node pos='0.1.4.0' tagname='img'/>
<node pos='0.1.4.1' tagname='br'/>
<node pos='0.1.4.2' tagname='i'/>
<node pos='0.1.4.2.0' tagname='font'/>
<node pos='0.1.4.2.1' tagname='a'/>
<node pos='0.1.4.2.2' tagname='font'/>
</ftree>
```

Das nächste Ergebnis sind die erhaltenen keywords mit ihren zugehörigen Konzepten und den Positionsangaben.

```
<LINGDATA>
<HEAD>hotel</HEAD>
<DOMAIN-TYPE>Hotel</DOMAIN-TYPE><POS>0.0.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.0.0</POS>
<HEAD>urlaub</HEAD>
<DOMAIN-TYPE>Urlaub</DOMAIN-TYPE><POS>0.0.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.0.0</POS>
<HEAD>hotel</HEAD>
<DOMAIN-TYPE>Hotel</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.1.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.1.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.2.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.2.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.3.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.3.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.4.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.4.0</POS>
<HEAD>Rostock</HEAD>
```

```
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.5.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.5.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.6.0</POS>
<HEAD>Mecklenburg</HEAD>
<DOMAIN-TYPE>Region</DOMAIN-TYPE><POS>0.1.1.0.0.0.0.0.0.6.0</POS>
<HEAD>hotel</HEAD>
<DOMAIN-TYPE>Hotel</DOMAIN-TYPE><POS>0.1.2.0.0.0.0.0.0</POS>
<HEAD>bild</HEAD>
<DOMAIN-TYPE>Bild</DOMAIN-TYPE><POS>0.1.2.0.0.0.0.0.0</POS>
<HEAD>hotel</HEAD>
<DOMAIN-TYPE>Hotel</DOMAIN-TYPE><POS>0.1.2.0.0.0.1.0.0.0.0</POS>
<HEAD>hotel</HEAD>
<DOMAIN-TYPE>Hotel</DOMAIN-TYPE><POS>0.1.2.0.0.0.2.0.0.0.0</POS>
<HEAD>bild</HEAD>
<DOMAIN-TYPE>Bild</DOMAIN-TYPE><POS>0.1.2.0.0.0.2.0.0.0.0</POS>
<HEAD>Rostock</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.3.0.0</POS>
<HEAD>tourist</HEAD>
<DOMAIN-TYPE>Tourist</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.5.1</POS>
<HEAD>bad</HEAD>
<DOMAIN-TYPE>Bad</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.6.1.0</POS>
<HEAD>telefon</HEAD>
<DOMAIN-TYPE>Telefon</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.6.1.0</POS>
<HEAD>tv</HEAD>
<DOMAIN-TYPE>Fernseher</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.6.1.0</POS>
<HEAD>radio</HEAD>
<DOMAIN-TYPE>Radio</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.8</POS>
<HEAD>minibar</HEAD>
<DOMAIN-TYPE>Minibar</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.8</POS>
<HEAD>einzelzimmer</HEAD>
<DOMAIN-TYPE>Einbettzimmer</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.11.1</POS>
<HEAD>doppelzimmer</HEAD>
<DOMAIN-TYPE>Zweibettzimmer</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.11.1</POS>
<HEAD>dreibettzimmer</HEAD>
<DOMAIN-TYPE>Mehrbettzimmer</DOMAIN-TYPE><POS>0.1.3.0.0.1.0.11.3</POS>
<HEAD>person</HEAD>
<DOMAIN-TYPE>Person</DOMAIN-TYPE><POS>0.1.3.0.0.3.0.1</POS>
<HEAD>haus</HEAD>
<DOMAIN-TYPE>Unterkunft</DOMAIN-TYPE><POS>0.1.3.0.0.3.0.1</POS>
<HEAD>Schwerin</HEAD>
<DOMAIN-TYPE>Stadt</DOMAIN-TYPE><POS>0.1.4.2.2</POS>
</LINGDATA>
```

Die letzte Darstellung beinhaltet die Zielstruktur mit dem resultierenden abstract. Weitere Daten, beispielsweise zur HTML–Seite, sind aufgrund der Speicherschnittstelle momentan nicht in der Zielstruktur enthalten. Solche Daten liegen nach dem Gathering vor und können gegebenenfalls hinzugefügt werden.

```
<list>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
```

```
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>krueger</name>
  </fragment>
  <constraint type='HEURISTIC'>MODIFIER</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
```

```
  </fragment><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment>
  <constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Stadt</type><inst>rostock</inst>
    <name>Rostock</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
```

```
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Mehrbettzimmer</type><inst>Mehrbettzimmer_1</inst>
    <name>dreibettzimmer</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
    <name>einzelzimmer</name>
  </fragment><fragment>
    <type>Fernseher</type><inst>Fernseher_1</inst>
    <name>tv</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Einbettzimmer</type><inst>Einbettzimmer_1</inst>
```

```
    <name>einzelzimmer</name>
  </fragment><fragment>
    <type>Telefon</type><inst>Telefon_1</inst>
    <name>telefon</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
</list>
```

Anhang D

Weitere HTML–Beispiele

Es werden hier weitere Beispiele von HTML–Seiten und deren resultierenden abstracts gezeigt.

Das erste Beispiel (Abbildung D.1) ist eine Zimmerbeschreibung des Strandhotels Hübner in Warnemünde auf dessen Informationsserver. Das Layout der Seite ist strukturell anders aufgebaut, als die Seiten von "www.all-in-all.de" (siehe zweites Beispiel und Beispiel aus Anhang A).

Das zweite Beispiel (Abbildung D.2) stammt vom HTML–Server "www.all-in-all.de" und beschreibt eine Übersicht über Hotels in und um Bad Doberan. Dabei sind einige Hotels durch einen Link mit einer konkreten Beschreibungseite verbunden. Die Liste der Hotels ist durch eine Tabelle dargestellt.

Nachfolgend das resultierende abstract der Seite "www.hotel-huebner.de/zimmer.html". Das Dokument wurde mit denselben Parametern, wie bei den "www.all-in-all.de"–Seiten verwendeten, analysiert. Auf der Web–Seite wurde das Konzept Zimmer als domänenspezifische Klasse herausgefunden. Der Hotelname wurde aufgrund der fehlenden Spezialgrammatik zur Eigennamenerkennung nicht erkannt, wobei zusätzliche Relationen, wie beispielsweise zwischen Hotel und Zimmer, dazugekommen wären. Auch ohne diese Information wurde der wesentliche Inhalt, und zwar die Zimmerbeschreibung, erkannt.

```
<list>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>info</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Foen</type><inst>Foen_1</inst>
```

```
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Dusche</type><inst>Dusche_1</inst>
    <name>dusche</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Zimmer</type><inst>Zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Zimmer</type><inst>Zimmer_1</inst>
    <name>einricht</name>
  </fragment>
  <constraint type='HEURISTIC'>MODIFIER</constraint>
</tuple>
<tuple name='getess-output' rel='IS_A'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>SAT_TV</type><inst>SAT_TV_2</inst>
    <name>satelliten-tv</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
```

```
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zimmer</type><inst>zimmer_1</inst>
    <name>zimmer</name>
  </fragment><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>information</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>information</name>
  </fragment><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>anderweitig</name>
  </fragment>
  <constraint type='HEURISTIC'>MODIFIER</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>information</name>
  </fragment><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>ausfuehrlich</name>
  </fragment>
  <constraint type='HEURISTIC'>MODIFIER</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Minibar</type><inst>Minibar_2</inst>
    <name>minibar</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_2</inst>
    <name>haustier</name>
  </fragment>
  <constraint type='HEURISTIC'>MODIFIER</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Information</type><inst>Information_2</inst>
    <name>info</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Radio</type><inst>Radio_1</inst>
    <name>radio</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
```

```
    <type>Foen</type><inst>Foen_1</inst>
    <name>foen</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Dusche</type><inst>Dusche_1</inst>
    <name>dusche</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Minibar</type><inst>Minibar_1</inst>
    <name>minibar</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Zweibettzimmer</type><inst>Zweibettzimmer_1</inst>
    <name>doppelzimmer</name>
  </fragment><fragment>
    <type>SAT_TV</type><inst>SAT_TV_2</inst>
    <name>satelliten-tv</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
</list>
```

Nachfolgend die resultierenden abstracts von “all-in-all.de/9172.htm”.

Das erste abstract entstand durch die Überschrift der Seite. Aufgrund der fehlenden Eigennamenerkennung bei der linguistischen Analyse wurde Bad von Bad Doberan nicht korrekt erkannt.

```
<list>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Pension</type><inst>Pension_1</inst>
    <name>pension</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple>
```

```
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple>
</list>
```

Die nächsten abstracts der Hotelübersichtsseite beschreiben jeweils ein im Dokument aufgelistetes Hotel. Die Hotels wurden aufgrund der fehlenden Spezialgrammatiken zu Eigennamen und Adressen nur in kleinen Datenstücken gefunden. Die Separierung in die einzelnen Teile funktioniert dagegen gut.

```
<list>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>landhotel</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>wittenbeck</name>
  </fragment>
  <constraint type='HEURISTIC'>(MODIFIER)</constraint>
</tuple>
</list>

<list>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Hotel</type><inst>Hotel_1</inst>
    <name>weid</name>
  </fragment>
  <constraint type='HEURISTIC'>(MODIFIER)</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>Mecklenburg</name>
  </fragment><fragment>
    <type>Region</type><inst>mecklenburg</inst>
    <name>haupt-G-S-trass</name>
  </fragment>
  <constraint type='HEURISTIC'>(MODIFIER)</constraint>
</list>

<list>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Hotel</type><inst>Hotel_2</inst>
    <name>hotel</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple>
```

```
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Gasthof</type><inst>Gasthof_1</inst>
    <name>gasthof</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
</list>

<list>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Pension</type><inst>Pension_1</inst>
    <name>pension</name>
  </fragment><fragment>
    <type>Strand</type><inst>Strand_1</inst>
    <name>strand</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Pension</type><inst>Pension_1</inst>
    <name>pension</name>
  </fragment><fragment>
    <type>Strand</type><inst>Strand_1</inst>
    <name>ostsee</name>
  </fragment>
  <constraint type='HEURISTIC'>NONE</constraint>
</tuple>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Strand</type><inst>Strand_1</inst>
    <name>strand</name>
  </fragment><fragment>
    <type>Strand</type><inst>Strand_1</inst>
    <name>ostsee</name>
  </fragment>
  <constraint type='HEURISTIC'>(MODIFIER)</constraint>
</tuple>
</list>

<list>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment><fragment>
    <type>Strand</type><inst>Strand_1</inst>
    <name>strand</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Unterkunft</type><inst>Unterkunft_2</inst>
    <name>gaestehaus</name>
  </fragment><fragment>
    <type>Strand</type><inst>Strand_1</inst>
    <name>strand</name>
  </fragment>
```

```
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple>
<tuple name='getess-output' rel='HAS_RELATION_WITH'><fragment>
    <type>Unterkunft</type><inst>Unterkunft_2</inst>
    <name>gaestehaus</name>
  </fragment><fragment>
    <type>Urlaub</type><inst>Urlaub_1</inst>
    <name>urlaub</name>
  </fragment>
  <constraint type='HEURISTIC'>(SENTENCE-HEURISTIC)</constraint>
</tuple>
</list>

<list>
<tuple name='getess-output' rel='EQUAL'><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>bad</name>
  </fragment><fragment>
    <type>Bad</type><inst>Bad_1</inst>
    <name>18208</name>
  </fragment>
  <constraint type='HEURISTIC'>(CARD)</constraint>
</tuple>
</list>
```

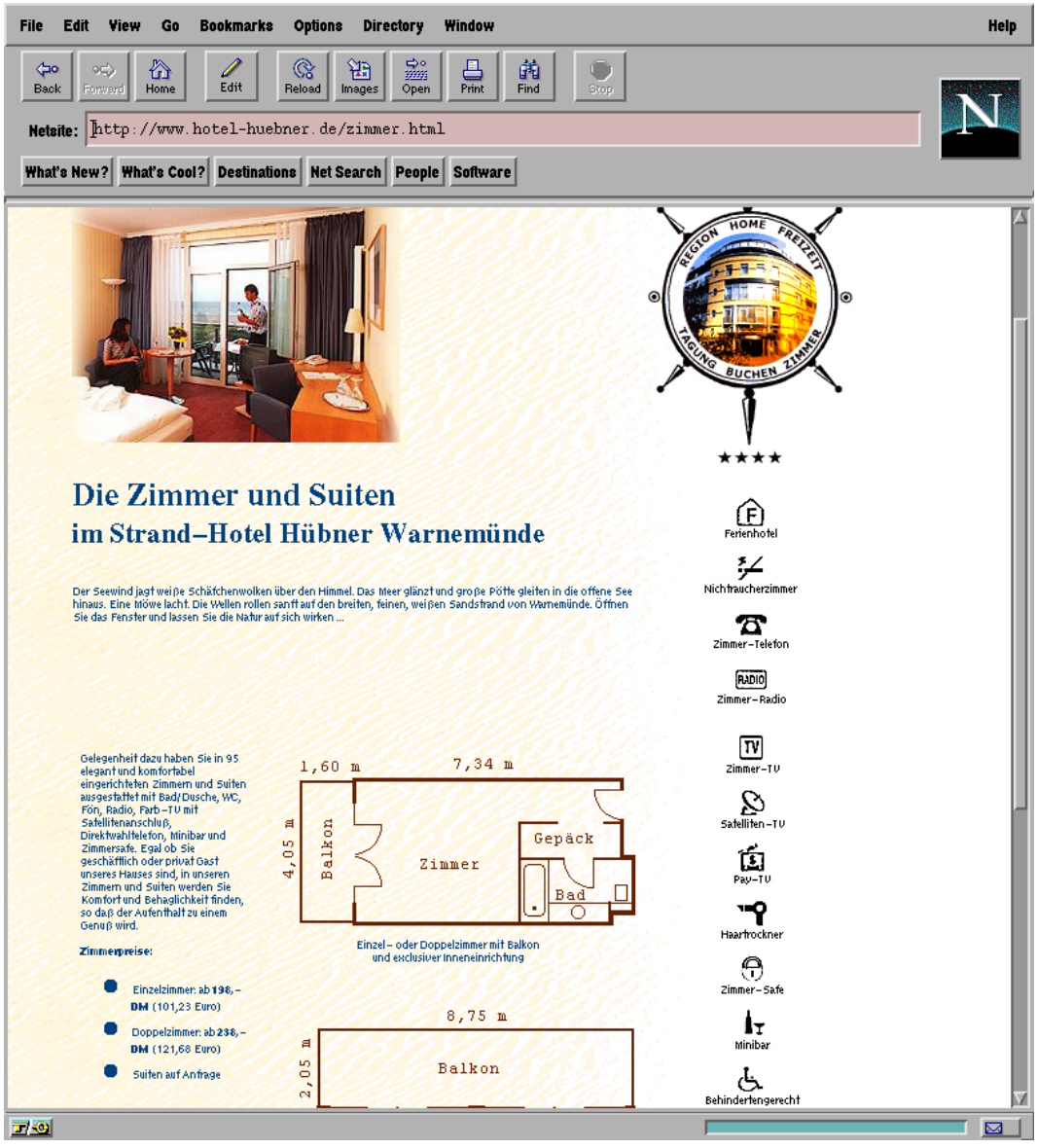

Abbildung D.1: HTML–Seite “zimmer.html” vom Server “www.hotel-huebner.de”

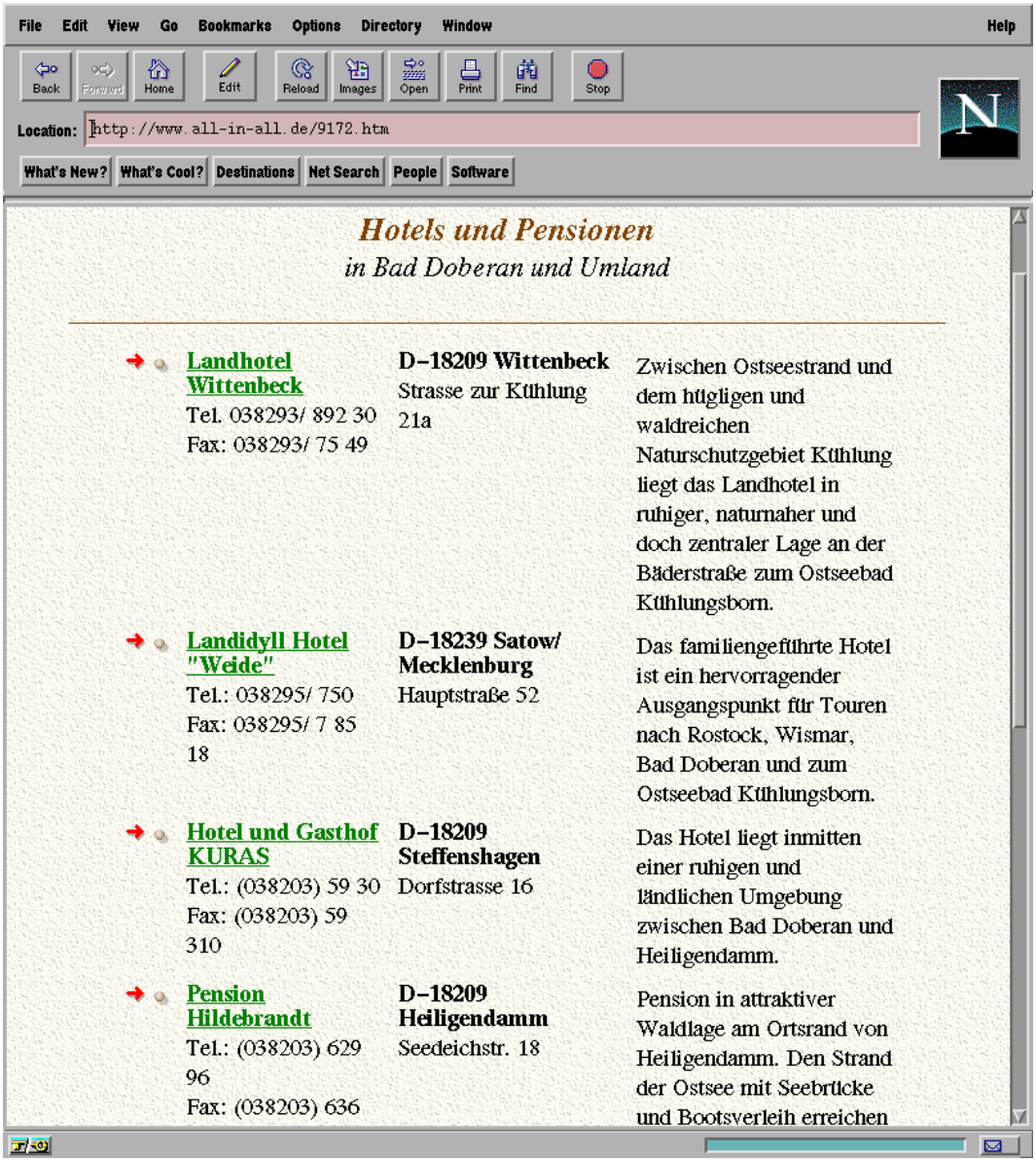

Abbildung D.2: HTML–Seite “9068.htm” vom Server “www.all-in-all.de”